

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**

(19)



Europäisches Patentamt

European Patent Office

Office européen des brevets



(11)

EP 0 726 004 B1

(12)

EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention
of the grant of the patent:

03.09.1997 Bulletin 1997/36

(21) Application number: **94914760.7**

(22) Date of filing: **11.04.1994**

(51) Int Cl.⁶: **H04L 29/06**

(86) International application number:
PCT/US94/03978

(87) International publication number:
WO 95/17062 (22.06.1995 Gazette 1995/26)

(54) OBJECT-ORIENTED RULE-BASED PROTOCOL SYSTEM

OBJEKTORIENTIERTES, AUF REGELN BASIERTES PROTOKOLLSYSTEM

SYSTEME DE PROTOCOLE A BASE DE REGLES, ORIENTE OBJETS

(84) Designated Contracting States:
DE FR GB

(30) Priority: **17.12.1993 US 169867**

(43) Date of publication of application:
14.08.1996 Bulletin 1996/33

(73) Proprietor: **TALIGENT, INC.**
Cupertino, CA 95014 (US)

(72) Inventor: **PETTUS, Christopher, E.**
San Francisco, CA 94114 (US)

(74) Representative: **Kindermann, Manfred**
Patentanwalt,
Sperberweg 29
71032 Böblingen (DE)

(56) References cited:

- **COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY.**, vol.35, no.7, July 1992, NEW YORK US pages 77 - 98 A.SINHA 'CLIENT-SERVER COMPUTING'
- **COMPUTER COMMUNICATION REVIEW.**, vol.21, no.4, September 1991, NEW YORK US pages 197 - 205, XP234937 C.TSCHUDIN 'FLEXIBLE PROTOCOL STACKS'

Note: Within nine months from the publication of the mention of the grant of the European patent, any person may give notice to the European Patent Office of opposition to the European patent granted. Notice of opposition shall be filed in a written reasoned statement. It shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

EP 0 726 004 B1

Description

Field Of The Invention

This invention relates, in general, to distributed computer networks and more specifically to rule-based protocol processing for distributed network directory and naming services.

Background Of The Invention

With the tremendous growth of data processing by means of independent, localized data processing devices, such as personal computers and mini computers, data networks have evolved to connect together physically-separated devices and to permit digital communication among the various devices connected to the network.

There are several types of networks, including local area networks (LANs) and wide area networks (WANs). A LAN is a limited area network and data devices connected to a LAN are generally located within the same building. The LAN typically consists of a transmission medium, such as a coaxial cable or a twisted pair which connects together various computers, servers, printers, modems and other digital devices. Each of the devices, which are collectively referred to as "nodes", is connected to the transmission medium at an address which uniquely identifies the node and is used to route data from one node to another. A node which provides resources and services is called a "server" node and a node which uses the resources and services is called a "client" node. A WAN generally encompasses a much larger area and may involve common carrier connections such as telephone lines.

The document, *Client-Server Computing*, by Alok Sinha, discloses the general concepts of client-server computing technology. This overview includes discussion of the basic paradigm, industry perspective, technology, connectivity interfaces, and future possibilities including database interfaces and graphical interfaces. This document is a reasonably detailed overview of client-server technology.

The document, *Flexible Protocol Stacks*, by C. Tchudin, speculates about the feasibility of flexible stacks and discloses a test system to "experiment with some fundamental problems which arise when protocol stacks are reconfigured dynamically." This document noted that "more research is needed to find stack composition techniques," and that more testing and debugging was needed. While this document considers the possibility of flexible stacks it admits that "the question remains if such a protocol environment can be realized," and that "[a]t the present state of [their] research not enough is known about the techniques required to handle arbitrary protocol stacks and the techniques needed to implement such an environment." Lastly, the document states "[b]efore one can hope to have such a general protocol

environment... more understanding about the management of running protocol stacks is needed."

LANs and WANs are often connected together in various configurations to form "enterprise" networks which may span different buildings or locations or extend across an entire continent. Enterprise networks are convenient for several reasons: they allow resource sharing - programs, data and equipment are available to all nodes connected to the network without regard to the physical location of the resource and the user. Enterprise networks may also provide reliability by making several redundant sources of data available. For example, important data files can be replicated on several storage devices so that, if one of the files is unavailable, for example, due to equipment failure, the duplicate files are available.

One of the most important characteristics of enterprise networks is that they have the capability of bringing a large and sophisticated set of services to all of the attached users for a reasonable cost. However, for the users to exploit the network potential, they must be able to identify, locate and access the network resources. When a network is small, locating and accessing the available services is relatively simple, but networks are growing larger and there are many networks that presently very large. Thousand node networks are common and million node networks are on the horizon.

An example of a very large network is the INTERNET network, which is used by some of the largest public and private organizations. Much of the power of this type of network goes unused simply because the users are either unaware of the facilities available to them or they find the methods of accessing the facilities difficult or confusing. Consequently, in order to assist users in locating and accessing network resources, many existing networks today utilize network directory or naming services which accept a resource identifier or name from a user and locate the network address that corresponds to the desired network resource.

For example, the entered identifier or name can be "descriptive" and specify a resource by describing enough of its attributes to distinguish it from other resources. Such descriptive names are most useful to human users who are searching the network for a resource that meets certain specified criteria, but they are also require the most computing resources and are often difficult to distribute effectively. There presently exist a number standards for such descriptive name services. For example, the Consultative Committee on International Telephony and Telegraphy (CCITT) and the International Standards Organization (ISO) have developed a standard for a descriptive name service known as X.500

Naming and directory services (these will be referred to together as "directory services" hereafter) are presently implemented in a variety of ways. The simplest implementation is to use a single, centralized database contained in a local server node to hold a list of

names and corresponding network addresses. An example of such a localized directory service is shown in Figure 1. Figure 1 illustrates a computer network arranged in a "client-server" configuration comprising a plurality of client nodes 106, 108, 120, 122 and 128 which may, for example, be workstations, personal computers, minicomputers or other computing devices on which run application programs that communicate over various network links including links 102, 110, 116, 126 and 136 with each other and with server nodes, such as nodes 100, 112, 124, 132 and 138. The server nodes may contain specialized hardware devices and software programs that can provide a service or set of services to all or some of the client nodes. The client nodes are the users of the various network services which, in turn, are provided by the server nodes.

Typically, the centralized directory service database 104 is located in one of the server nodes, such as node 100. A client node, such as client node 108, can access the directory service by connecting to server node 100, entering a resource identifier or name and retrieving the network address of the associated service. By means of conventional database techniques, a client node may be able to search over the database in order to locate a given resource. In addition, many directory services support browsing by using partial name descriptions, "wild cards" and placeholders. Such centralized directory services with single databases work well in small networks where the number of network addresses is small. However, in larger networks, it is often not feasible to store all the resource identifiers in one central location. Further, a single database represents a single point of failure which can disable the entire network. In addition, a centralized database often suffers from poor performance. For example, while it may be relatively efficient for a local client, such as client 108, to connect to server 100 and access database 104, a remote client, such as client 120, which must link through several servers, 124 and 112, along with a "gateway" link 116, will incur a significant amount of network overhead and the overall system "cost" of the access will be high. With a large number of remote access attempts, directory service provider 100 can quickly become both a processing and communication bottleneck for the entire network.

In order to overcome these problems, additional prior art techniques have been developed which distribute the database data over multiple locations. Such a system is shown schematically in Figure 2. Figure 2 depicts a client-server type of network which is similar to that shown in Figure 1. In particular, elements which correspond in the two figures have corresponding numeral designations. For example, client 108 in Figure 1 is similar to client 208 in Figure 2. The difference between the two networks is that the directory service database has been replicated in a number of the server nodes. For example, server node 200 contains a directory service database 204 as do server nodes 212 (database 214), server node 232 (database 230) and server node 224

(database 218). There are a number of prior art methods for replicating the data in each of the databases. Some systems replicate each resource identifier individually in each database, other systems replicate the entire database. Still other systems replicate individual nodes or limit replication by partitioning the database in some manner.

The distributed system shown in Figure 2 avoids the problems associated with the centralized database. Since the data is replicated, there is no single point of failure and, since the data is usually available on a nearby server node, there are no "remote" client nodes and network overhead is greatly reduced.

However, the distributed system has its own problems. For example, some method must be used to insure data consistency if multiple sources can update the databases. Some systems force data consistency by keeping all copies of the data tightly synchronized in a manner similar to a conventional database system. Other systems insure data integrity by means of conventional concurrency arbitration schemes.

Such distributed naming and directory services are effective on homogeneous networks in which the same access methods and protocols apply over the entire network. In this case, a consistent set of names and rules can be developed to permit location and access of various resources with relative ease. However, many large networks are heterogeneous - not only do the networks comprise many types of different computers, including workstations, personal computers, mini-computers, super-computers and main frames, but the network itself is often composed of many independent smaller networks which are connected together by interfaces called "gateways". These smaller networks may have their own access methods and protocols. Further, the heterogeneous construction and organization of these large networks does not lend itself to central control and management which could dictate common methods and protocols.

In many large networks which are comprised of a set of smaller networks which are connected together, each of the underlying separate networks may have its own different directory service utilizing a specific protocol. In this type of network a user may have to be familiar with each network directory service protocol and may have to shift from protocol to protocol as searches are performed from network to network. Consequently, in such a heterogeneous network, one of the main difficulties in accessing network resources arises from a lack of a consistent globally-accessible directory of network resources which can operate over heterogeneous networks without involving the user in the details and the protocol involved in accessing each of these separate networks.

Today's networking services have various protocol systems. In the prior art, applications must contain detailed information pertaining to protocol architectures for networking. This requirement mandated application ties

to particular protocols. This prevented dynamic configuration of networks because applications could not support protocol changes without a source code change.

Accordingly, it is an object of the present invention to allow applications to specify a type and quality of service.

Summary Of The Invention

The foregoing problems are solved and the foregoing objects are achieved in one illustrative embodiment of the invention in which a rule based protocol selection mechanism selects the appropriate protocol configuration based on the type and quality of service. For each protocol family, a set of rules is defined to translate the type and quality of service into a particular set of common protocols.

Brief Description Of The Drawings

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings, in which:

Figure 1 is a block schematic diagram of a prior art client server network which incorporates a local directory service.

Figure 2 is a block schematic diagram of a prior art client server network which incorporates a distributed directory server.

Figure 3 is a block schematic diagram of a computer system, for example, a personal computer system in which the rule based protocol selection mechanism operates.

Figure 4 is a block schematic diagram of a client server network which incorporates the rule based protocol selection mechanism.

Figure 5 is a detailed block schematic diagram of a prior art protocol stack used to transmit data between two nodes structure in accordance with the International Standards Organization seven layer model.

Figure 6 is a block schematic diagram of the major components of the communications directory service.

Figure 7 is a schematic diagram of an illustrative directory tree set up which allows browsing over various directory services and other network services.

Figure 8 is a block schematic diagram of the main components of a server node illustrating how a service program interacts with the communications directory service.

Figure 9 is a simplified flowchart of the steps involved in making a new service available on the network.

Figure 10 is an expanded flowchart of the steps carried out by the service program in order to activate a service object.

Figure 11 is a block schematic diagram of the main components of a client node illustrating how an applica-

tion program interacts with the communications directory service to access a service.

Figure 12 is a simplified flowchart of the steps involved in accessing a service available on the network.

Figure 13 is an expanded flowchart of the steps carried out by the service program in order to activate a service object.

Detailed Description Of Illustrative Embodiments

The invention is preferably practiced in the context of an operating system resident on a personal computer such as the IBM™ PS/2™ or Apple™ Macintosh™ computer. A representative hardware environment is depicted in Figure 3, which illustrates a typical hardware configuration of a computer 300 in accordance with the subject invention. The computer 300 is controlled by a central processing unit 302, which may be a conventional microprocessor; a number of other units, all interconnected via a system bus 308, are provided to accomplish specific tasks. Although a particular computer may only have some of the units illustrated in Figure 3 or may have additional components not shown, most computers will include at least the units shown.

Specifically, computer 300 shown in Figure 3 includes a random access memory (RAM) 306 for temporary storage of information, a read only memory (ROM) 304 for permanent storage of the computer's configuration and basic operating commands and an input/output (I/O) adapter 310 for connecting peripheral devices such as a disk unit 313 and printer 314 to the bus 308, via cables 315 and 312, respectively. A user interface adapter 316 is also provided for connecting input devices, such as a keyboard 320, and other known interface devices including mice, speakers and microphones to the bus 308. Visual output is provided by a display adapter 318 which connects the bus 308 to a display device 322 such as a video monitor. The workstation has resident thereon and is controlled and coordinated by operating system software such as the Apple System/7™ operating system.

In a preferred embodiment, the invention is implemented in the C++ programming language using object-oriented programming techniques. C++ is a compiled language, that is, programs are written in a human-readable script and this script is then provided to another program called a compiler which generates a machine-readable numeric code that can be loaded into, and directly executed by, a computer. As described below, the C++ language has certain characteristics which allow a software developer to easily use programs written by others while still providing a great deal of control over the reuse of programs to prevent their destruction or improper use. The C++ language is well-known and many articles and texts are available which describe the language in detail. In addition, C++ compilers are commercially available from several vendors including Borland International, Inc. and Microsoft Corporation. Accord-

ingly, for reasons of clarity, the details of the C++ language and the operation of the C++ compiler will not be discussed further in detail herein.

As will be understood by those skilled in the art, Object-Oriented Programming (OOP) techniques involve the definition, creation, use and destruction of "objects". These objects are software entities comprising data elements and routines, or functions, which manipulate the data elements. The data and related functions are treated by the software as an entity and can be created, used and deleted as if they were a single item. Together, the data and functions enable objects to model virtually any real-world entity in terms of its characteristics, which can be represented by the data elements, and its behavior, which can be represented by its data manipulation functions. In this way, objects can model concrete things like people and computers, and they can also model abstract concepts like numbers or geometrical designs.

Objects are defined by creating "classes" which are not objects themselves, but which act as templates that instruct the compiler how to construct the actual object. A class may, for example, specify the number and type of data variables and the steps involved in the functions which manipulate the data. An object is actually created in the program by means of a special function called a constructor which uses the corresponding class definition and additional information, such as arguments provided during object creation, to construct the object. Likewise objects are destroyed by a special function called a destructor. Objects may be used by using their data and invoking their functions.

The principle benefits of object-oriented programming techniques arise out of three basic principles; encapsulation, polymorphism and inheritance. More specifically, objects can be designed to hide, or encapsulate, all, or a portion of, the internal data structure and the internal functions. More particularly, during program design, a program developer can define objects in which all or some of the data variables and all or some of the related functions are considered "private" or for use only by the object itself. Other data or functions can be declared "public" or available for use by other programs. Access to the private variables by other programs can be controlled by defining public functions for an object which access the object's private data. The public functions form a controlled and consistent interface between the private data and the "outside" world. Any attempt to write program code which directly accesses the private variables causes the compiler to generate an error during program compilation which error stops the compilation process and prevents the program from being run.

Polymorphism is a concept which allows objects and functions which have the same overall format, but which work with different data, to function differently in order to produce consistent results. For example, an addition function may be defined as variable A plus variable B (A+B) and this same format can be used whether the A and B are numbers, characters or dollars and

cents. However, the actual program code which performs the addition may differ widely depending on the type of variables that comprise A and B. Polymorphism allows three separate function definitions to be written, one for each type of variable (numbers, characters and dollars). After the functions have been defined, a program can later refer to the addition function by its common format (A+B) and, during compilation, the C++ compiler will determine which of the three functions is actually being used by examining the variable types. The compiler will then substitute the proper function code. Polymorphism allows similar functions which produce analogous results to be "grouped" in the program source code to produce a more logical and clear program flow.

The third principle which underlies object-oriented programming is inheritance, which allows program developers to easily reuse pre-existing programs and to avoid creating software from scratch. The principle of inheritance allows a software developer to declare classes (and the objects which are later created from them) as related. Specifically, classes may be designated as subclasses of other base classes. A subclass "inherits" and has access to all of the public functions of its base classes just as if these functions appeared in the subclass. Alternatively, a subclass can override some or all of its inherited functions or may modify some or all of its inherited functions merely by defining a new function with the same form (overriding or modification does not alter the function in the base class, but merely modifies the use of the function in the subclass). The creation of a new subclass which has some of the functionality (with selective modification) of another class allows software developers to easily customize existing code to meet their particular needs.

Although object-oriented programming offers significant improvements over other programming concepts, program development still requires significant outlays of time and effort, especially if no pre-existing software programs are available for modification. Consequently, a prior art approach has been to provide a program developer with a set of pre-defined, interconnected classes which create a set of objects and additional miscellaneous routines that are all directed to performing commonly-encountered tasks in a particular environment. Such pre-defined classes and libraries are typically called "application frameworks" and essentially provide a pre-fabricated structure for a working application.

For example, an application framework for a user interface might provide a set of pre-defined graphic interface objects which create windows, scroll bars, menus, etc. and provide the support and "default" behavior for these graphic interface objects. Since application frameworks are based on object-oriented techniques, the pre-defined classes can be used as base classes and the built-in default behavior can be inherited by developer-defined subclasses and either modified or overridden to allow developers to extend the framework and

create customized solutions in a particular area of experience. This object-oriented approach provides a major advantage over traditional programming since the programmer is not changing the original program, but rather extending the capabilities of the original program. In addition, developers are not blindly working through layers of code because the framework provides architectural guidance and modeling and, at the same time, frees the developers to supply specific actions unique to the problem domain.

There are many kinds of application frameworks available, depending on the level of the system involved and the kind of problem to be solved. The types of frameworks range from high-level application frameworks that assist in developing a user interface, to lower-level frameworks that provide basic system software services such as communications, printing, file systems support, graphics, etc. Commercial examples of application frameworks include MacApp (Apple), Bedrock (Symantec), OWL (Borland), NeXT Step App Kit (NeXT), and Smalltalk-80 MVC (ParcPlace).

While the application framework approach utilizes all the principles of encapsulation, polymorphism, and inheritance in the object layer, and is a substantial improvement over other programming techniques, there are difficulties which arise. These difficulties are caused by the fact that it is easy for developers to reuse their own objects, but it is difficult for the developers to use objects generated by other programs. Further, application frameworks generally consist of one or more object "layers" on top of a monolithic operating system and even with the flexibility of the object layer, it is still often necessary to directly interact with the underlying operating system by means of awkward procedural calls.

In the same way that an application framework provides the developer with prefabricated functionality for an application program, a system framework, such as that included in a preferred embodiment, can provide a prefabricated functionality for system level services which developers can modify or override to create customized solutions, thereby avoiding the awkward procedural calls necessary with the prior art application frameworks programs. For example, consider a printing framework which could provide the foundation for automated pagination, preprint processing and page composition of printable information generated by an application program. An application software developer who needed these capabilities would ordinarily have to write specific routines to provide them. To do this with a framework, the developer only needs to supply the characteristics and behavior of the finished output, while the framework provides the actual routines which perform the tasks.

A preferred embodiment takes the concept of frameworks and applies it throughout the entire system, including the application and the operating system. For the commercial or corporate developer, systems integrator, or OEM, this means all of the advantages that have been illustrated for a framework such as MacApp

can be leveraged not only at the application level for such things as text and user interfaces, but also at the system level, for services such as printing, graphics, multi-media, file systems, I/O, timing, etc.

Figure 4 shows a schematic overview of the illustrative client-server network illustrated in Figures 1 and 2 which has now been provided with the communications directory service. A comparison of Figures 2 and 4 indicates that, although the general network configuration is the same for both networks, when the communications directory service is used as shown in Figure 4, all of the nodes now include a communications directory service (CDS) module. In particular, server nodes 400, 412, 424, 432 and 438 now include the communications directory service modules 405, 413, 425, 430 and 439, respectively. In addition, any or all server nodes may also include a conventional directory service 404, 414, 418 and 430, respectively. These conventional directory services will be referred to as physical directory services hereinafter to distinguish them from the communications directory service.

In addition to the server nodes, each of the client nodes 406, 408, 420, 422 and 428 (CDS 407, 409, 421, 423 and 429, respectively) also includes a communications directory service. As will hereinafter be explained in detail, in enterprise networks such as that shown in Figure 4, information to be sent from one node to another is generally divided into discrete messages or "packets" and the packets are transmitted between nodes in accordance with a predefined "protocol". In this context a "protocol" consists of a set of rules or procedures defining how the separate nodes are supposed to interact with each other.

In order to reduce design complexity, most networks are organized as a series of "layers" or "levels" so that information passing from one node to another is transmitted from layer to layer. Within each layer, predetermined services or operations are performed and the layers communicate with each other by means of predefined protocols. The purpose for the layered design is to allow a given layer to offer selected services to other layers by means of a standardized interface while shielding those layers from the details of actual implementation within the layer. As will hereinafter be explained in detail, both the client and server nodes cooperate with the communications directory service modules in order to structure the network layers which, in turn, control the type and parameters of the connections between client and servers in accordance with the type of service being offered.

In an attempt to standardize network architectures (the overall name for the sets of layers and protocols used within a network), a generalized model has been proposed by the International Standards Organization (ISO) as a first step towards international standardization of the various protocols now in use. The model is called the open systems interconnection (OSI) reference model because it deals with the interconnection of

systems that are "open" for communication with other systems. The proposed OSI model has seven layers which are termed (in the order which they interface with each other) the "physical", "data link", "network", "transport", "session", "presentation" and "application" layers. The purpose of the OSI model is to attempt to standardize the processes conducted within each layer.

In accordance with the OSI model, the processes carried out in the physical layer are concerned with the transmission of raw data bits over a communication channel. The processes carried out in the data link layer manipulate the raw data bit stream and transform it into a data stream that appears free of transmission errors. The latter task is accomplished by breaking the transmitted data into data frames and transmitting the frames sequentially accompanied with error correcting mechanisms for detecting or correcting errors.

The network layer processes determine how data packets are routed from the data source to the data destination by selecting one of many alternative paths through the network. The function of the transport layer processes is to accept a data stream from a session layer, split it up into smaller units (if necessary), pass these smaller units to the network layer, and to provide appropriate mechanisms to ensure that the units all arrive correctly at the destination, with no sequencing errors, duplicates or missing data.

The session layer processes allow users on different machines to establish "sessions" or "dialogues" between themselves. A session allows ordinary data transport between the communicating nodes, but also provides enhanced services in some applications, such as dialogue control, token management and synchronization. The presentation layer performs certain common functions that are requested sufficiently often to warrant finding a general solution for them, for example, encoding data into a standard format, performing encryption and decryption and other functions. Finally, the application protocol layer contains a variety of protocols that are commonly needed, such as database access, file transfer, etc.

For example, Figure 5 is a diagram of a prior art protocol stack used to connect two nodes in accordance with the OSI standard seven-layer architecture. In accordance with the OSI standard, each protocol stack for a node consists of seven layers. For example, stack 528 for Client node 408 comprises an application layer 500, a presentation layer 502, a session layer 504, a transport layer 506, a network layer 508, a data link layer 510 and a physical layer 512. The operation and the purpose of each of these layers has been previously discussed. Similarly, stack 532 for Server node 432 consists of layers 514-526. Although the actual data communication occurs between the two physical layers 512 and 526 over a data link 530, when the stack is arranged as shown in Figure 5, each layer can be thought of as communicating with its "peer" which is a layer at the same level as a given layer. For example, the application lay-

ers 500 and 514 can be thought of as communicating directly even though information passes through all of the layers 502-512, across data link 530 and back through the layers 516-526. Similarly presentation layers 502 and 516 can communicate peer-to-peer.

The protocol stacks such as those shown in Figure 5 are typically implemented using a plurality of data buffers. Each data buffer constitutes a protocol layer in which processing is done. After processing is complete in a layer the data may be transferred to another buffer for further processing in accordance with another layer.

In accordance with one aspect of the invention, the protocol stacks which control peer-to-peer communications in the network are configured by stack definitions stored in the communications directory service. These stack definitions are associated with each service type available on the network so that the protocol stacks can be dynamically configured in response to service requests from the application programs.

A more detailed diagram of a communications directory service module is shown in detail in Figure 6; the module includes three major components: a hierarchical directory tree 602 which allows each of the physical directory services and other services provided by the network to be located by means of a conventional tree searching techniques; a set of stack definition objects 604 which are used to program the dynamically reconfigurable stacks that allow a client to request data from a remote server over a prespecified communication link and a set of "service objects" 606. Each service object is associated with one service available on the network and contains the network address or exchange address at which the service is available and a reference 608 to one or more of the stack definition objects 604. As will hereinafter be explained in detail, a reference to one of these service objects is obtained by an application program desiring to access the corresponding service. The information in the object identified by this reference is then sent to the reconfigurable stack in order to set up the communication path.

The stack definitions 604 each consist of a set of layer definitions that specify the processing carried out in each layer and the interactions between the layers. The stack definitions are defined at the time that the communications links are installed on the network system. In particular a stack definition is provided for each different type of communication link on the system.

The communications directory service 600 has a number of other features which make its operation particularly convenient for users on the network. In particular, the directory service stores information regarding the available services and directory services in the directory tree as a series of objects. In addition, the stored libraries in the communications directory service contain objects and, thus, both application programs and service programs can communicate directly with the communications directory service by means of objects. Since the communications directory service is present

on each of the nodes, a client need not connect with any directory service, but rather using the directory service, instead the objects themselves provide the interface. Further, the use of predefined objects, such as the service objects 606, provides a simple method to allow an application program to open communications with a desired service. The service objects can be sent directly to the networking service in order to open a communications path without requiring the application program to concern itself with the details of the actual path construction. In addition, because any existing physical directory services are encapsulated in the node objects of the directory tree 602, the physical directory services are completely hidden from the client application allowing the client application to interact with the communications directory service with a consistent protocol and feature set.

A more detailed diagram of the directory tree 602 found in the communications directory service is shown in Figure 7. As will hereinafter be explained, the directory tree can be traversed by means of simple search commands and full browsing capabilities are provided by using wild cards and placeholders. The directory tree is designed so that objects are returned as the result of searches with the type of object which is returned being determined by the implementation of the portion of the directory which returns the object. Examples of objects which can be returned from a directory tree search are references to the service objects 606 shown in Figure 6 which contain information used to connect to remote service; principle objects which contain security and authentication information about users; "business cards" which contain collections of information about users and other classes which may be added to the directory tree by program developers. The directory tree is organized as a single hierarchical tree such as that shown in Figure 7. It should be noted that the tree configuration shown in Figure 7 is for an illustrative purposes only and an actual tree configuration can differ significantly from the illustrated configuration without departing from the scope and principles of the present invention. Each of the nodes in the tree is formed by a "namespace" object. A namespace object can refer to other namespace objects or can refer directly to services or other physical directory services. The ultimate root of the directory tree is the root namespace object 700; the root namespace object 700 and the other namespace objects which form the nodes of the tree are inserted into a conventional tree structure which can be traversed to find the node members.

Each node object, such as root namespace object 700, includes methods which facilitate directory searches. In particular, these methods may include a search method which returns an iterator over all entries in the directory, a method which returns entries to which the namespace refers and a method which returns entries that match a given property expression. Additional methods may be provided to obtain the root namespace

from lower level directory node.

In the tree directory shown in Figure 7, the ultimate nodes or "leaf" nodes are the physical directory services and the other available network services. As these leaf nodes are also namespace objects, they may include methods which return the associated directory protocol and methods which can interact with the physical directory to search or browse the directory. Since each of the leaf nodes contains methods which are written specifically for the associated service, the methods deal with the protocol issues involved with the associated service allowing the user to interface with the communications directory service in a consistent manner no matter what directory service is involved.

For example, in Figure 7, three nodes 704, 706 and 708 are shown which interact with three separate physical directory services. A fourth node, 710, is also provided, which node is called a "native" namespace node. This latter node contains a reference to each of the services that are provided by the network. As will hereinafter be explained, in order to make each service available to the users, it is "registered" in the native namespace 710. "Registration" in the namespace means to insert a reference to the service into the directory where it can be found by someone traversing the directory. Essentially, registration consists of streaming the associated service object into the native namespace where it can be discovered by queries.

In addition, to registering a service, it is also possible to "publish" a service in the directory or in the physical directory services resident in the leaf nodes. Publication differs from registration in that publication involves inserting a limited set of information about the service into a directory node. This limited set of information may, for example, consist of the service name, type of service and a connection address. In a preferred form of the invention, the native namespace handles publications - when a service entity is registered with the native namespace, the native namespace determines in which other namespaces the entity should be published and directs the publication accordingly.

In order to provide full branching capabilities, intermediate namespaces, such as namespace 702 may be inserted between the root namespace 700 and the leaf nodes 704-710. The intermediate namespace refers to other namespaces. As previously mentioned, the inventive communications directory service interacts with both service programs and application programs to transparently set up the necessary communication links between the client and server. This interaction involves the creation of service objects in the communications directory service by the service providers and the configuration of the protocol stacks in the server nodes. A client desiring to access a service retrieves the associated service object and uses it to configure the protocol stacks in the client node to set up the communication link. Figures 8-10 describe the operations performed by a service program operating in a server node in order to

make a new service available on the network for use by application programs running in the client nodes. Figure 8 is a schematic block diagram of the portions of the server node programs that are involved in the creation and activation of a new service. Figure 9 is an illustrative flow chart which describes the interaction between the service program, the communications directory service and the node networking services in order to establish the new service and to configure the networking service to operate over an appropriate communication link. Figure 10 is a more detailed block diagram illustrating the activation of the new service by activating the associated service object. More particularly, Figure 8 illustrates the basic configuration of a server node in providing a service to a remote client. The server node is arranged with a common area, or system address space, 800 which address space would include operating system programs and various shared libraries that are used by the service applications running on the system. In particular, the system address space 800 includes the communications directory service 804 and the networking service programs and libraries 816. Also in the server node is the service program which runs in its own address space 810. As will be hereinafter explained, the service program 806 interacts with the communications directory service 804 to create a service object which then resides in the communications directory service 804. This service object is distributed to all of the other nodes in this system and, thus, is available on a local basis to all of the clients on the network. Because the service object includes the appropriate stack definitions for configuring the networking service 816, the application program together with the communications directory service can set up the communications path using the previously stored definitions without involving a client application program in the construction details.

Referring to Figure 9, the creation of a new service begins in step 900 and proceeds to step 902. In step 902, the developer of the service program enables the creation of a new service object containing configuration parameters which are appropriate to the new service. This may be done by subclassing the service object classes located in the directory service. In particular, in order to create a new service object class, the service program developer specifies a unique name for the service, the type of service and, optionally, various communications link types which can be used to access the service. In accordance with normal object-oriented programming language operation, this subclassing information is included in the service program code during compilation. Therefore, when the service program 806 is installed in the service program address space 810 in the appropriate server node, the constructor of the service object subclass can be called to construct a service object in the communications directory service 804 as illustrated in step 904 (Figure 9). During the process of calling the constructor, the type and quality of service information is passed to the communications directory

service as schematically indicated by arrow 802

As previously mentioned, the communications directory service 804 includes a set of stack definitions in shared libraries. These stack definitions are created when the communications links are defined and are associated with a particular transport mechanism. The stack definitions each consist of a set of layer definitions. The layer definitions control the processing of the data in each layer and the interactions between layers. Each stack definition completely defines the stack from the transport layer through the physical layer (these layers are described in detail above).

In the process of creating a new service object, the communications directory service 804 uses the type and quality of service information provided by the service program to construct a session layer and include appropriate references to the stored communications stack definitions as set forth in step 906. If more than one communication link can be used to access the new service, appropriate stack definitions are referenced in the new service object and the session layer is constructed in order to select one of the communication links based on various criteria, such as the quality of service desired and the availability of the communication link.

Thus, the newly-created service object contains a session layer and the appropriate stack definitions which define the stack from the transport layer to the physical layer. The service object is stored in the communications directory service. However, at this time, no service address or exchange address is provided in the service object because the service object, although created, is not active. Thus, a service object can be created any point when the service program is installed in the server node, but the service program however does not become activated until further steps are taken as described below.

More particularly, in order to activate the new service, the service program instructs the communications directory service 804 to send the stored service object to the dynamically reconfigurable protocol stack (DRPS) 822 in networking service 816 as described in step 908. The manner in which the service object is transferred to the DRPS is illustrated in Figure 8. More particularly, the service program 806 first creates a service program interface 828 in its own address space 810. The communication between the service program and the service program interface is schematically indicated by arrow 808. The service program interface 828, in turn, creates a configuration data stream 814 which can stream data to a server interface 818 which is permanently available and located in the networking service 816.

The service program 806 then retrieves the service object including the network configuration data. The configuration data passes from the communications directory service to the service program interface 828 as indicated by arrow 812. The configuration data is then streamed to the system address space 800 as indicated by arrow 814.

Next, as illustrated in step 910, the service program activates the service object which, in turn, causes the service exchange address to be returned to the communications directory service 804. A more detailed description of the activation sequence is illustrated in the flow chart shown in Figure 10. More particularly, the activation sequence starts in step 1000 and proceeds to step 1002. In step 1002, the communications directory service 804 makes the service available by publishing the service on any underlying physical directory services if this is appropriate. This publication consists of registering the name of the service and, in some cases, the service address in the underlying directory services.

The routine next proceeds to step 1004 in which the service is registered with the communications directory service. As previously mentioned, such registration involves streaming the service object into the native namespace node of the communications directory service. At this point, a reference to the service is added to the CDS directory tree so that the service can be located by user traversing the tree.

Next, in step 1006, the stack definition references which have been passed to the server interface 818 in networking service 816 by the service program 806 are used to set up the stack definitions that will be used to reconfigure the DRPS 822. In step 1008, the created stack definitions are passed to the DRPS (this operation is shown schematically by arrow 820). At this time the information in the service object is also used to set up the session layer 823.

In step 1010, the exchange address, or the network address, of the session layer 823 in the DRPS 822 is returned to the communications directory service 804. In particular, the networking service 816 obtains the session layer address from the DRPS 822 when the session layer is set up. This address is returned, via the service interface 818, configuration data stream 814, to the service program interface 828. Finally, the exchange address is returned, via the data stream 812, to the communications directory service 804. The exchange address, as previously mentioned, is stored in the associated service object located in the communications directory service 804 so that it can later be retrieved by a client program. At this point, activation is complete and the activation routine shown in Figure 10 ends in step 1012. In step 912, the exchange address is returned to the communications directory service and the service activation routine ends in step 914.

A separate data stream is also set up by the service program at this time so that, at a subsequent time, when a client node requests service from the server node, the incoming service requests can be received. In particular, service request data arriving over the physical communication link 824 is passed through the configured DRPS 822 via a separate request data stream 826 which links the session layer 823 to the service program interface 828. The service program interface 828, in turn, forwards the request, via data stream 808, to the service

program 806. Reply information is returned by service program 806, via data stream 808, service program interface 828, request data stream 826, DRPS 822, and physical communication link 824 to the client node.

After a new service has been added to the local communication directory service, the copies on the other nodes must be updated. This updating is carried out in a conventional fashion. For example, it is possible to periodically distribute copies on removable disks to all nodes. However, a more preferred method of distributing copies of the communications directory service would be for the local node which received the update to broadcast the update information to all other nodes. In order to accomplish this broadcast, the broadcast message includes a special header which causes all of the protocol stacks to be set to a predetermined default protocol. In this manner the broadcast message can be received at all nodes.

Figures 11 through 13 illustrate the steps involved when a client application coordinates with the communications directory service to access a remote service. In particular, Figure 11 illustrates the appropriate sections of the client node which are involved in accessing a remote service. As in the server node, the client node includes a system address space 1110 which, in turn, contains the communications directory service 1112 and a networking service 1118. The application program 1100 runs in its own application address space 1104. The interactions of the application program 1100, the directory service 1112 and the networking service 1118 are described in more detail in the flow charts set forth in Figures 12 and 13.

More particularly, the client service access routine starts in step 1200 and proceeds to step 1202. As indicated in step 1202, the client interacts with the communications directory service to get a reference to one of the service objects stored in the communications directory service. This interaction is shown schematically by arrow 1106 on Figure 11. As previously mentioned, this interaction may involve a user directly if the user searches over the directory tree located in the communications directory service 1112. Alternatively, this interaction may involve an intervening application program which cooperates with the communications directory service to select a service in a visual manner, such as, by dragging a document icon onto a service icon.

In any case, in accordance with step 1204, a reference to the service object identified by the communications directory service 1112 is returned to the application program 1100 as shown schematically by arrow 1108. The application program, in turn, creates a client interface object 1126 in preparation for sending the configuration data to the network service 1118. The service object reference is passed by the communications directory service 1112, via a configuration data stream 1114, to the client interface 1126. From the client interface 1126 the configuration data is streamed over configuration data stream 1116 to a server interface object 1120

located in the networking service 1118. The service interface object 1120 is created when the networking service 1118 is created during system boot up and is permanently resident in the networking service.

In accordance with step 1206, application program 1100 then activates the service object reference. Figure 13 indicates, in more detail, the steps involved in activating the service object reference. More particularly, the activation routine starts in step 1300 and proceeds to step 1302. In step 1302, the service object reference is resolved in any of the underlying physical directory services, if appropriate. This resolution is performed by using the service name located in the service object to search over the underlying directory services and to obtain the network address. Alternatively, if the service reference is registered in the native namespace of the communications directory service 1112, then the service address or the service exchange can be obtained directly from the service object reference.

In step 1304, the stack definitions contained in the service object are used by the server interface 1120 and the networking service 1118 to set up protocol stack layers for configuring the DRPS 1124. Next, in step 1306, the created stack definitions are passed to the DRPS as indicated by arrow 1122. These stack definitions then set up DRPS 1124 and configure the communication link in preparation for sending request and reply data between the application program 1100 and the remote service (not shown in Figure 11).

Next, in step 1308, the address of the session layer 1123 in the DRPS 1124 is returned to the server interface 1120. The activation routine then finishes in step 1310. Returning to step 1208 of Figure 12, the server interface 1120 exchanges the address of the session layer 1123 for the remote service exchange obtained from the service object reference and returns the remote service exchange (as indicated in step 1208), via configuration data stream 1116, client interface 1126 and data path 1102 to the application program 1100. Thus, when the application program communicates with the remote service, it uses the remote service address passed through the communications directory service to the networking service.

As set forth in step 1210, a separate data path is set up to send service requests from application program 1100 to the remote service. This separate data path comprises data path 1102, client interface 1126 and the session layer 1123 of the DRPS 1124. In accordance with step 1212, the request information then sent out over physical communication link 1130 to the remote service location. Reply information returns via DRPS 1124, data stream 1128, client interface 1126 and data path 1102 to the application program 1100. The service request routine then finishes in step 1214.

Claims

1. A multi-node computer network system for connecting a client node (406, 408, 420, 422, 428) to a server node (400, 412, 424, 432, 438) over a plurality of different communication links (316, 315, 366), said computer network system comprising: a plurality of nodes, at least one processor per node (302); a memory (304, 306, 313) attached to said at least one processor and under said processor's control; said computer network system characterized by:

- (a) said plurality of different communication links differentiated by a type of service and a quality of service (906), said type and said quality of service stored in said memory;
- (b) a plurality of protocols for transmitting data between said client node and said server node across said communication links;
- (c) said memory comprising a first program logic (1100) and a second program logic (1110), said first program logic for requesting a connection to said server node, said connection dependent on said type and said quality of service; and said second program logic for identifying at least one of said plurality of protocols and at least one of said plurality of links based on said type and quality of service.

2. A client node of a client-server system said client-server system comprising a distributed computer network, said network comprising remote procedure call (RPC) services (1112, 804, 806), said RPC services accessed through a RPC protocol, said client node interconnected with a server node via a communications medium (824, 1130) to form said client-server system; said client node comprising: a processor (302); a memory (304, 306, 313) accessible by said processor; a first program logic (1126) for processing a plurality of service packets, said first program logic contained within an application program (1100), said application program resident in said memory; a network adapter for transmitting and receiving said packets over said communications medium; and a second program logic (1118) resident in said memory for transferring said packets to and from said adapter; said client node characterized by:

- (a) a third program logic resident in said memory for controlling said processor, said third program logic being an object-oriented operating system;
- (b) a caller object (1106) created by said application program from data and functions stored in said operating system, said second program logic responsive to said caller object, to encapsulate said packets in accordance with said

- RPC protocol to enable said first program logic to call an associated service (836) residing in a dispatcher object of said server node;
- (c) a dynamically-configurable protocol stack (1124) comprising a plurality of vertically-linked protocol layer objects, said protocol stack configured to interact with said adapter to transfer said packets over said communication medium via said adapter; and
- (d) a remote stream object (1128) for establishing synchronous data stream transactions between said application program and said protocol stack, said remote stream object and protocol stack cooperating to complete a communications data path within said client node.
3. A client node as recited in claim 2, wherein said plurality of vertically-linked protocol layer objects include upper protocol layer objects (500, 502) that are unique to said application program executing in said client node and lower protocol layer objects (504, 506, 508, 510, 512) that are shared among other application programs executing in said client node.
 4. A client node as recited in claim 3, wherein said upper protocol layers include an application layer object (500) for exchanging said packets with said application program.
 5. A client node as recited in claim 4, wherein said upper protocol layers include a presentation layer object (502) for presenting said packets in a predetermined format to said lower protocol layer objects.
 6. A client node as recited in claim 5, wherein said application layer and presentation layer objects reside in a process address space (1104) of said client node.
 7. A client node as recited in claim 6, wherein said caller object and said application program further reside in said process address space.
 8. A client node as recited in claim 7, wherein said lower protocol layer objects and said operating system reside in a system address space (1110) of said client node.
 9. A client node as recited in claim 8, wherein said remote stream object includes a fourth program logic to ensure a consistent format for presentation of said packets between said process address space and said system address space.
 10. A client node as recited in claim 9, wherein said remote stream object comprises one of a request/reply model object (1128) for establishing short-term synchronous transactions between said client and server nodes and a partial remote operation service element model object for binding said nodes over long-term synchronous transactions.
 11. A method for connecting a client node (406, 408, 420, 422, 428) to a server node (400, 412, 424, 432, 438) within a multi-node computer network system over a plurality of different communication links (316, 315, 366), said plurality of links connecting a plurality of nodes; each of said plurality of nodes comprising at least one processor (302); said method comprising the steps of:
 - (a) associating a link type of service and a link quality of service with each of said plurality of links (802, 902, 904);
 - (b) specifying a desired type of service and a desired quality of service (1206);
 - (c) identifying at least one protocol and at least one link based on said desired type of service and desired quality of service from said link type of service and link quality of service (1304); and
 - (d) establishing a connection between said client node and said server node using said at least one protocol and said at least one link (1210).
 12. A method for connecting a client node to a server node within a client-server system said client-server system comprising a distributed computer network, said network comprising remote procedure call (RPC) services (1112, 804, 806), said RPC services accessed using a RPC protocol, said client node interconnected with said server node via a communications medium (824, 1130) to form said client-server system; said client node comprising: a processor (302); a memory (304, 306, 313) accessible by said processor; a first program logic (1126) for processing a plurality of service packets, said first program logic contained within an application program (1100), said application program resident in said memory; a network adapter for transmitting and receiving said packets over said communications medium; and a second program logic (1118) resident in said memory for transferring said packets to and from said adapter; an object-oriented operating system (1110) for controlling said processor, said operating system comprising a third program logic and said operating system resident in said memory; said method comprising the steps of:
 - (a) constructing a caller object (1202) by said first program logic from data and functions contained within said operating system;
 - (b) presenting said caller object to said second program logic (2104);
 - (c) said second program logic responding to

- said presentation of said caller object by encapsulating said packets in accordance with said RPC protocol and enabling said first program logic to call an associated service (836) residing in a dispatcher object of said server node;
- (d) constructing a dynamically-configurable protocol stack (1124) from a plurality of vertically-linked protocol layer objects, said protocol stack configured to interact with said adapter to transfer said packets over said communication medium via said adapter; and
- (e) constructing a remote stream object (1128) establishing synchronous data stream transactions between said application program and said protocol stack, said remote stream object and said protocol stack cooperating to complete a communications data path within said client node.
13. A method as recited in claim 12, wherein constructing said dynamically-configurable protocol stack comprises:
- (d1) assembling upper protocol layer objects that are unique to said application program executing in said client node; and
- (d2) assembling lower protocol layer objects that are shared among other application programs executing in said client node.
14. A method as recited in claim 13, wherein assembling said upper protocol layer objects further comprises including an application layer object for exchanging said packets with said application program.
15. A method as recited in claim 14, wherein assembling said upper protocol layer objects further comprises including a presentation layer object for presenting said packets in a predetermined format to said lower protocol layer objects.
16. A method as recited in claim 15 wherein said application layer object, said presentation layer objects said caller object and said application program reside in said process address space of said client node, and said lower protocol layer objects and said operating system reside in a system address space of said client node;
- said method further comprising said remote stream object ensuring a consistent format for said presentation of said packets between said process address space and said system address space.
17. A method as recited in claim 16, wherein said remote stream object comprises one of a request/reply model object for establishing short-term synchronous transactions between said client and

server nodes and a partial remote operation service element model object for binding said nodes over long-term synchronous transactions.

Patentansprüche

- Ein Vielfachknoten-Computer-Netzwerkssystem zum Verbinden eines Client-Netzknotens (406, 408, 420, 422, 428) mit einem Server-Netzknoten (400, 412, 424, 432, 438) über eine Vielzahl von unterschiedlichen Kommunikationsverbindungen (316, 315, 366), besagtes Computer-Netzwerkssystem enthält eine Vielzahl von Netzknoten, wenigstens einen Prozessor pro Netzknoten (302), einen Speicher (304, 306, 313), der an besagten wenigstens einen Prozessor angeschlossen ist und von diesem gesteuert wird, besagtes Computer-Netzwerkssystem ist gekennzeichnet durch:
 - besagte Vielzahl von unterschiedlichen Kommunikationsverbindungen ist unterschieden nach einem Servicetyp und einer Servicequalität (906), besagter Typ und besagte Qualität sind in besagtem Speicher gespeichert;
 - eine Vielzahl von Protokollen zur Übertragung von Daten zwischen besagtem Client-Netzknoten und dem besagten Server-Netzknoten über besagte Kommunikationsverbindungen;
 - besagter Speicher enthält eine erste Programmlogik (1100) und eine zweite Programmlogik (1110), besagte erste Programmlogik dient zur Anforderung einer Verbindung zu besagtem Server-Netzknoten, besagte Verbindung ist abhängig von besagtem Typ und besagter Qualität des Services; und besagte zweite Programmlogik dient zur Identifizierung von wenigstens einem der besagten Vielzahl von Protokollen und wenigstens einer der besagten Vielzahl von Verbindungen auf der Basis von besagtem Typ und besagter Qualität des Services.
- Ein Client-Netzknoten eines Client-Server-Systems, das ein verteiltes Computer-Netzwerk enthält, besagtes Netzwerk umfaßt Prozedur-Fernauf-ruf(RPC)-Dienste (1112, 804, 806), auf besagte RPC Dienste wird durch ein RPC Protokoll zugegriffen, besagter Client-Netzknoten ist mit einem Server-Netzknoten über ein Kommunikationsmedium (824, 1130) verbunden zur Bildung des besagten Client-Server-Systems; besagter Client-Netzknoten enthält einen Prozessor (302); einen Speicher (304, 306, 313), auf den durch besagten Prozessor zugreifbar ist; eine erste Programmlogik

- (1126) zur Verarbeitung einer Vielzahl von Servicepaketen, besagt erst Programmlogik ist in einem Anwendungsprogramm (1100) enthalten, das in besagtem Speicherresident ist; in einem Netzwerkadapter zum Aussenden und Empfangen besagter Pakete über besagtes Kommunikationsmedium; und eine zweite Programmlogik (1118), die in besagtem Speicherresident ist zum Übertragen besagter Pakete zu und von besagtem Adapter; besagter Client-Netzknoten ist gekennzeichnet durch:
- (a) eine dritte Programmlogik, die in besagtem Speicherresident ist zur Steuerung des besagten Prozessors, besagte dritte Programmlogik ist ein objektorientiertes Betriebssystem;
 - (b) ein Aufrufobjekt (1106), das durch das besagte Anwendungsprogramm aus in besagtem Betriebssystem gespeicherten Daten und Funktionen erzeugt wird, besagte zweite Programmlogik spricht auf besagtes Aufrufobjekt an zur Einkapselung besagter Pakete in Übereinstimmung mit besagtem RPC Protokoll, um die besagte erste Programmlogik in die Lage zu versetzen, einen verbundenen Service (836) aufzurufen, der in einem Dispatcher-Objekt des besagten Server-Netzknotens enthalten ist;
 - (c) einen dynamisch konfigurierbaren Protokollstapel (1124), der eine Vielzahl von vertikal verbundenen Protokollschicht-Objekten enthält, besagter Protokollstapel ist konfiguriert, mit besagtem Adapter zusammenzuwirken zum Übertragen besagter Pakete über besagtes Kommunikationsmedium durch besagten Adapter; und
 - (d) ein Fern-Strömungs-Objekt (1128) zum Herstellen synchroner Datenstrom-Transaktionen zwischen besagtem Anwendungsprogramm und besagtem Protokollstapel, besagtes Fern-Strömungs-Objekt und besagter Protokollstapel kooperieren zur Vervollständigung eines Kommunikationsdatenpfades innerhalb des besagten Client-Netzknotens.
3. Ein Client-Netzknoten nach Anspruch 2, worin besagte Vielzahl von vertikal verbundenen Protokollschicht-Objekten obere Protokollschicht-Objekte (500, 502) enthalten, die allein dem besagten Anwendungsprogramm zugeordnet sind, das in besagtem Client-Netzknoten ausgeführt wird, und untere Protokollschicht-Objekte (504, 506, 508, 510, 512) enthalten, in die sich andere Anwendungsprogramme teilen, die in besagtem Client-Netzknoten ausgeführt werden.
 4. Ein Client-Netzknoten nach Anspruch 3, worin besagte obere Protokollschicht ein Anwendungs-schicht-Objekt (500) enthalten zum Austausch besagter Pakete mit dem besagten Anwendungsprogramm.
 5. Ein Client-Netzknoten nach Anspruch 4, worin besagte obere Protokollschichten ein Präsentationsschicht-Objekt (502) enthalten zur Präsentation besagter Pakete den besagten unteren Protokollschicht-Objekten in einem vorgegebenen Format.
 6. Ein Client-Netzknoten nach Anspruch 5, worin sich besagtes Anwendungsschicht-Objekt und besagtes Präsentationsschicht-Objekt in einem Prozeßadressraum (1104) des besagten Client-Netzknotens gespeichert sind.
 7. Ein Client-Netzknoten nach Anspruch 6, worin besagtes Aufrufobjekt und besagtes Anwendungsprogramm ebenfalls in besagtem Prozeßadressraum enthalten sind.
 8. Ein Client-Netzknoten nach Anspruch 7, worin besagte untere Protokollschicht-Objekte und besagtes Betriebssystem in einem Systemadressraum des besagten Client-Netzknotens enthalten sind.
 9. Ein Client-Netzknoten nach Anspruch 8, worin besagtes Fern-Strömungs-Objekt eine vierte Programmlogik enthält zur Sicherung eines konsistenten Formats zur Darstellung von besagten Paketen zwischen besagtem Prozeßadressraum und besagtem Systemadressraum.
 10. Ein Client-Netzknoten nach Anspruch 9, worin besagtes Fern-Strömungs-Objekt ein Anforderung/Antwort-Modell-Objekt (1128) enthält zur Ausführung von kurzzeitigen synchronen Transaktionen zwischen besagten Client- und Server-Netzknoten und ein Teil-Fern-Operations-Service-Element-Modell-Objekt zum Verbinden der besagten Netzknoten für synchrone Langzeit-Transaktionen.
 11. Ein Verfahren zur Kopplung eines Client-Netzknotens (406, 408, 420, 422, 428) mit einem Server-Netzknoten (400, 412, 424, 432, 438) in einem Vielfachknoten-Computer-Netzwerkssystem über eine Vielzahl von unterschiedlichen Kommunikationsverbindungen (316, 315, 366), besagte Verbindungen verbinden eine Vielzahl von Netzknoten; jeder der besagten Vielzahl von Netzknoten enthält wenigstens einen Prozessor (302); besagtes Verfahren umfaßt die Schritte:
 - (a) Verknüpfen eines Verbindung-Servicetyps und einer Verbindung-Servicequalität mit jeder der besagten Vielzahl von Verbindungen (802,

902, 904);

(b) Angeben eines gewünschten Servicetyps und in r g gewünschten Servicequalität (1206);

5

(c) Identifizieren von wenigstens einem Protokoll und von wenigstens einer Verknüpfung, die auf dem besagten gewünschten Servicetyp und der besagten gewünschten Servicequalität von besagtem Verbindung-Servicetyp und besagter Verbindung-Servicequalität (1304) beruht; und

10

(d) Herstellen einer Kopplung zwischen besagten Client-Netzknoten und besagten Server-Netzknoten unter Verwendung des besagten wenigstens einem Protokolls und der besagten wenigstens einer Verbindung (1210).

15

12. Ein Verfahren zur Kopplung eines Client-Netzknoten mit einem Server-Netzknoten in einem Client-Server-Systems, das ein verteiltes Computer-Netzwerk enthält, besagtes Netzwerk umfaßt Prozeduren-Fernauf (RPC) Dienste (1112, 804, 806), auf besagte RPC Dienste wird durch ein RPC Protokoll zugegriffen, besagter Client-Netzknoten ist mit einem Server-Netzknoten über ein Kommunikationsmedium (824, 1130) verbunden zur Bildung des besagten Client-Server-Systems; besagter Client-Netzknoten enthält einen Prozessor (302); einen Speicher (304, 306, 313), der durch besagten Prozessor zugreifbar ist; eine erste Programmlogik (1126) zur Verarbeitung einer Vielzahl von Servicepaketen, besagte erste Programmlogik ist in einem Anwendungsprogramm (1100) enthalten, das in besagtem Speicher resident ist; einen Netzwerkadapter zum Aussenden und Empfangen besagter Pakete über besagtes Kommunikationsmedium; und eine zweite Programmlogik (1118), die in besagtem Speicher resident ist zum Übertragen besagter Pakete zu und von besagtem Adapter; ein objektorientiertes Betriebssystem, das eine dritte Programmlogik umfaßt und das in besagtem Speicher enthalten ist;
- besagtes Verfahren umfaßt die Schritte:

20

25

30

35

40

45

(a) Erzeugen eines Aufrufobjekts (1202) durch besagte erste Programmlogik aus in besagtem Betriebssystem enthaltenen Daten und Funktionen;

50

(b) Präsentieren des besagten Aufrufobjekts der besagten zweiten Programmlogik (2104);

(c) Antworten durch besagte zweite Programmlogik auf das besagte Präsentieren des besagten Aufrufobjekts durch Einkapselung besagter Paket in Übereinstimmung mit besagtem RPC

55

Protokoll und Aktivieren besagter ersten Programmlogik zum Aufruf eines verbundenen Services (836), der in einem Dispatch r-Objekt des besagten Server-Netzknotens enthalten ist;

(d) Erzeugen eines dynamisch konfigurierbaren Protokollstapels (1124) aus einer Vielzahl von vertikal verbundenen Protokollschicht-Objekten enthält, besagter Protokollstapel ist konfiguriert, mit besagtem Adapter zusammenzuwirken zum Übertragen besagter Pakete über besagtes Kommunikationsmedium durch besagten Adapter; und

(e) Erzeugen eines Fern-Strömungs-Objekts (1128) zum Herstellen synchroner Datenstrom-Transaktionen zwischen besagtem Anwendungsprogramm und besagtem Protokollstapel, besagtes Fern-Strömungs-Objekt und besagter Protokollstapel kooperieren zur Vervollständigung eines Kommunikationsdatenpfades innerhalb des besagten Client-Netzknotens.

13. Ein Verfahren nach Anspruch 12, worin die Erzeugung eines dynamisch konfigurierbaren Protokollstapels umfaßt:

(d1) Assemblieren von oberen Protokollschicht-Objekten, die allein dem besagten Anwendungsprogramm zugeordnet sind, das in besagtem Client-Netzknoten ausgeführt wird; und

(d2) Assemblieren von unteren Protokollschicht-Objekten, in die sich andere Anwendungsprogramme teilen, die in besagtem Client-Netzknoten ausgeführt werden.

14. Ein Verfahren nach Anspruch 13, worin das Assemblieren besagter oberer Protokollschicht-Objekte die Einbeziehung eines Anwendungsschicht-Objekts umfaßt zum Austausch besagter Pakete mit besagtem Anwendungsprogramm.

15. Ein Verfahren nach Anspruch 14, worin das Assemblieren besagter oberer Protokollschichten die Einbeziehung eines Präsentationsschicht-Objekts umfaßt zur Präsentation besagter Pakete den besagten unteren Protokollschicht-Objekten in einem vorgegebenen Format.

16. Ein Verfahren nach Anspruch 15, worin sich besagtes Anwendungsschicht-Objekt, besagte Präsentationsschicht-Objekte, besagtes Aufrufobjekt und besagtes Anwendungsprogramm in besagtem Prozeßadressraum des besagten Client-Netzknotens

enthalten sind, und besagte untere Protokollschicht-Objekte und besagtes Betriebssystem in einem Systemadressraum des besagten Client-Netzknötens enthalten sind;
 besagtes Fern-Strömungs-Objekt ist in konsistentem Format sich zur Präsentation von besagten Paketen zwischen besagtem Prozeßadressraum und besagtem Systemadressraum.

17. Ein Verfahren nach Anspruch 16, worin besagtes Fern-Strömungs-Objekt ein Anforderung/Antwort-Modell-Objekt (1128) enthält zur Ausführung von kurzzeitigen synchronen Transaktionen zwischen besagten Client- und Server-Netzknötens und ein Teil-Fern-Operations-Service-Element-Modell-Objekt zum Verbinden der besagten Netzknötens für synchrone Langzeit-Transaktionen.

Revendications

1. Système de réseau d'ordinateurs multi-noeud pour connecter un noeud client (406, 408, 420, 422, 428) à un noeud serveur (400, 412, 424, 432, 438) par une pluralité de liaisons de communication différentes (316, 315, 366), ledit système de réseau d'ordinateurs comprenant une pluralité de noeuds, au moins un processeur par noeud (302), une mémoire (304, 306, 313) connectée audit processeur et sous le contrôle du processeur,
 ledit système de réseau d'ordinateurs étant caractérisé par :
 - a) ladite pluralité de liaisons de communication différentes est différenciée par un type de service et une qualité de service (906), lesdits type et qualité de service étant emmagasinés dans ladite mémoire,
 - b) une pluralité de protocoles pour transmettre des données entre un noeud client et ledit noeud serveur au moyen desdites liaisons de communication,
 - c) ladite mémoire comprend une première logique programme (1100) et une seconde logique programme (1110), ladite première logique programme demandant une connexion audit noeud serveur, ladite connexion dépendant dudit type et de ladite qualité de service, et ladite seconde logique programme identifiant au moins une parmi ladite pluralité de protocoles et au moins une parmi ladite pluralité de liaisons basés sur ledit type et ladite qualité de service.
2. Noeud client d'un système serveur de clients, ledit système comprenant un réseau d'ordinateurs distribué, ledit réseau comprenant des services d'appel (RPC) de procédure éloignés (1112, 804, 806) dont l'accès se fait par un protocole RPC, ledit

noeud client étant interconnecté avec un noeud serveur via un support de communication (824, 1130) pour former ledit système serveur de clients, ledit noeud client comprenant : un processeur (302), une mémoire (304, 306, 313) accessible par ledit processeur, une première logique de programme (1126) pour traiter une pluralité de paquets de service, ladite logique de programme étant contenue dans un programme d'application (1100), ledit programme d'application résidant en mémoire, un adaptateur de réseau pour transmettre et recevoir lesdits paquets sur ledit support de communication, et une seconde logique de programme (1118) résidant dans ladite mémoire pour transférer lesdits paquets au et à partir dudit adaptateur, ledit noeud client étant caractérisé par :

- a) une troisième logique de programme résidant dans ladite mémoire pour contrôler ledit processeur, ladite troisième logique de programme étant un système d'exploitation orienté objet,
- b) un objet d'appel (1106) créé par ledit programme d'application à partir des données et des fonctions emmagasinées dans ledit système d'exploitation, ladite seconde logique de programme étant sensible audit objet d'appel pour encapsuler lesdits paquets conformément audit protocole RPC pour permettre à ladite première logique de programme d'appeler un service associé (836) résidant dans un objet de distribution dudit noeud serveur,
- c) une pile de protocoles pouvant être configurée de façon dynamique (1124) comprenant une pluralité d'objets de couches de protocole liés verticalement, ladite pile étant configurée pour interagir avec ledit adaptateur pour transférer lesdits paquets sur ledit support de communication via ledit adaptateur, et
- d) un objet de flot éloigné (1128) pour établir des transactions par séquences de données synchronisées entre ledit programme d'application et ladite pile de protocoles, ledit objet de flot et ladite pile de protocoles coopérant pour compléter un chemin de données de communication à l'intérieur dudit noeud client.

3. Noeud client tel que défini dans la revendication 2, dans lequel ladite pluralité d'objets de couches de protocole liés verticalement comprend des objets de couches de protocole supérieures (500, 502) qui sont uniques pour ledit programme d'application s'exécutant dans ledit noeud client et des objets de couches de protocole inférieures (504, 506, 508, 510, 512) qui sont partagés par d'autres programmes d'application s'exécutant dans ledit noeud client.

4. Noeud client tel que défini dans la revendication 3, dans lequel les couches de protocole supérieures comprennent un objet de couche d'application (500) pour échanger lesdits paquets avec ledit programme d'application. 5
5. Noeud client tel que défini dans la revendication 4, dans lequel les couches de protocole supérieures comprennent un objet de couche de présentation (502) pour présenter lesdits paquets dans un format prédéterminé auxdits objets de couches de protocole inférieures. 10
6. Noeud client tel que défini dans la revendication 5, dans lequel lesdits objets de couche d'application et de couche de présentation résident dans un espace adresse de procédés (1104) dudit noeud client. 15
7. Noeud client tel que défini dans la revendication 6, dans lequel ledit objet d'appel et ledit programme d'application résident en outre dans ledit espace adresse de procédé. 20
8. Noeud client tel que défini dans la revendication 7, dans lequel lesdits objets de couches de protocole inférieures et ledit système d'exploitation résident dans un espace adresse de système (1110) dudit noeud client. 25
9. Noeud client tel que défini dans la revendication 8, dans lequel ledit objet de flot éloigné comprend une quatrième logique de programme pour assurer un format consistant pour la présentation desdits paquets entre ledit espace adresse de procédé et ledit espace adresse de système. 30 35
10. Noeud client tel que défini dans la revendication 9, dans lequel ledit objet de flot éloigné comprend un des objets modèles de requête/réponse (1128) pour établir des transactions synchrones court terme entre lesdits noeuds client et serveur et un objet modèle d'élément de service d'opération éloignée partielle pour lier lesdits noeuds sur des transactions synchrones long terme. 40 45
11. Méthode pour connecter un noeud client (406, 408, 420, 422, 428) à un noeud serveur (400, 412, 424, 432, 438) dans un système de réseau d'ordinateurs multi-noeud,) par une pluralité de liaisons de communication différentes (316, 315, 366), ladite pluralité de communications connectant une pluralité de noeuds, chacun de ladite pluralité de noeuds comprenant au moins un processeur (302), ladite méthode comprenant les étapes de : 50 55
- a) associer un type de service de liaison et une qualité de service de liaison à chacune d ladite

pluralité de liaisons (802, 902, 904),
 b) spécifier un type de service désiré et une qualité de service désirée,
 c) identifier au moins un protocole et au moins une liaison basée sur ledit type de service désiré et ladite qualité de service désirée à partir dudit type de service de liaison et de ladite qualité de service de liaison, (1304), et
 d) établir une connexion entre ledit noeud client et ledit noeud serveur en utilisant ledit protocole et ladite liaison (1210).

12. Méthode pour connecter un noeud client à un noeud serveur dans un système serveur de clients, ledit système serveur de clients comprenant un réseau d'ordinateurs distribué, ledit réseau comprenant des services d'appel (RPC) de procédure éloignées (1112, 804, 806) lesdits services RPC étant accédés en utilisant un protocole RPC, ledit noeud client étant interconnecté avec un noeud serveur via un support de communication (824, 1130) pour former ledit système serveur de clients, ledit noeud client comprenant: un processeur (302), une mémoire (304, 306, 313) accessible par ledit processeur, une première logique de programme (1126) pour traiter une pluralité de paquets de service, ladite logique de programme étant contenue dans un programme d'application (1100), ledit programme d'application résidant en mémoire, un adaptateur de réseau pour transmettre et recevoir lesdits paquets sur ledit support de communication, et une seconde logique de programme (1118) résidant dans ladite mémoire pour transférer lesdits paquets au et à partir dudit adaptateur, un système d'exploitation orienté objet (1110) pour contrôler ledit processeur, ledit système d'exploitation comprenant une troisième logique de programme et ledit système d'exploitation résidant dans ladite mémoire ;
 ladite méthode comprenant les étapes de :

a) construire un objet d'appel (1202) par ladite première logique de programme à partir des données et des fonctions emmagasinées dans ledit système d'exploitation,
 b) présenter ledit objet d'appel à la seconde logique de programme (2104),
 c) ladite seconde logique de programme répondant à la présentation dudit objet d'appel en encapsulant lesdits paquets conformément audit protocole RPC pour permettre à ladite première logique de programme d'appeler un service associé (836) résidant dans un objet de distribution dudit noeud serveur,
 d) construire une pile de protocoles pouvant être configurée de façon dynamique (1124) à partir d'une pluralité d'objets de couches de protocole liés verticalement, ladite pile de protocoles étant configurée pour interagir avec l -

dit adaptateur pour transférer lesdits paquets sur ledit support de communication via ledit adaptateur, et

synchrones long term .

e) construire un objet de flot éloigné (1128) établissant des transactions par séquences de données synchronisées entre ledit programme d'application et ladite pile de protocoles, ledit objet de flot et ladite pile de protocoles coopérant pour compléter un chemin de données de communication à l'intérieur dudit noeud client.

13. Méthode telle que définie dans la revendication 12, dans laquelle ladite pile de protocoles pouvant être configurée de façon dynamique comprend :

d1) l'assemblage d'objets de couches de protocole supérieures qui sont uniques pour ledit programme d'application s'exécutant dans ledit noeud client et

d2) l'assemblage d'objets de couches de protocole inférieures qui sont partagés par d'autres programmes d'application s'exécutant dans ledit noeud client.

14. Méthode telle que définie dans la revendication 13, dans laquelle l'assemblage desdits objets de couches de protocole supérieures comprend l'inclusion d'un objet de couche d'application pour échanger lesdits paquets avec ledit programme d'application.

15. Méthode telle que définie dans la revendication 14, dans laquelle l'assemblage desdits objets de couches de protocole supérieures comprend l'inclusion d'un objet de couche de présentation pour présenter lesdits paquets dans un format prédéterminé auxdits objets de couches de protocole inférieures.

16. Méthode telle que définie dans la revendication 15, dans laquelle ledit objet de couche d'application, lesdits objets de couche de présentation et ledit programme d'application résident dans un espace adresse de procédés dudit noeud client, et lesdits objets de couches de protocole inférieures et ledit système d'exploitation résident dans un espace adresse de système dudit noeud client,

ladite méthode comprenant en outre ledit objet de flot éloigné assurant un format consistant pour la présentation desdits paquets entre ledit espace adresse de procédé et ledit espace adresse de système.

17. Méthode telle que définie dans la revendication 16, dans laquelle ledit objet de flot éloigné comprend un des objets modèles de requête/réponse pour établir des transactions synchrones court terme entre lesdits noeuds client et serveur et un objet modèle d'élément de service d'opération éloignée partielle pour lier lesdits noeuds sur des transactions

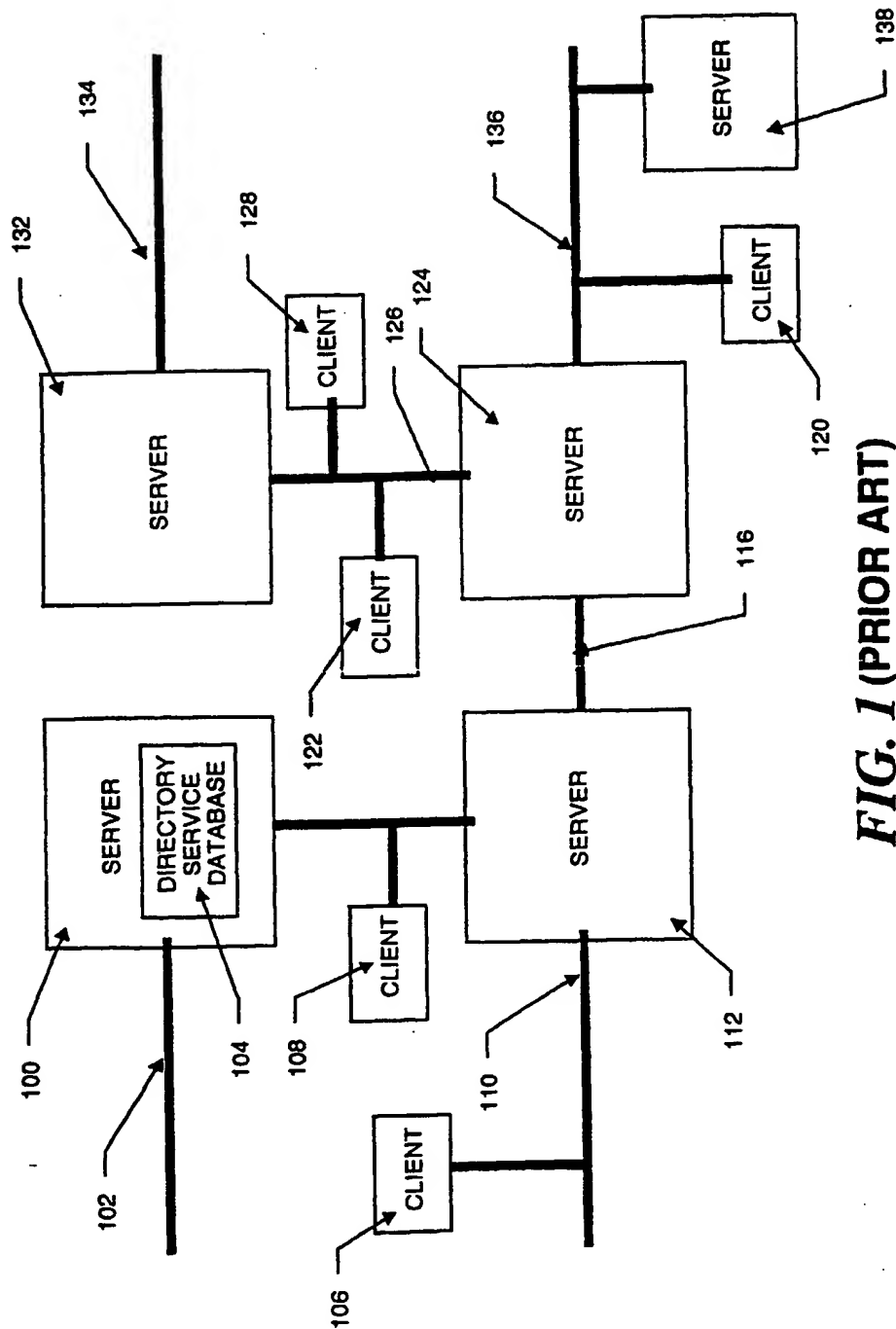


FIG. 1 (PRIOR ART)

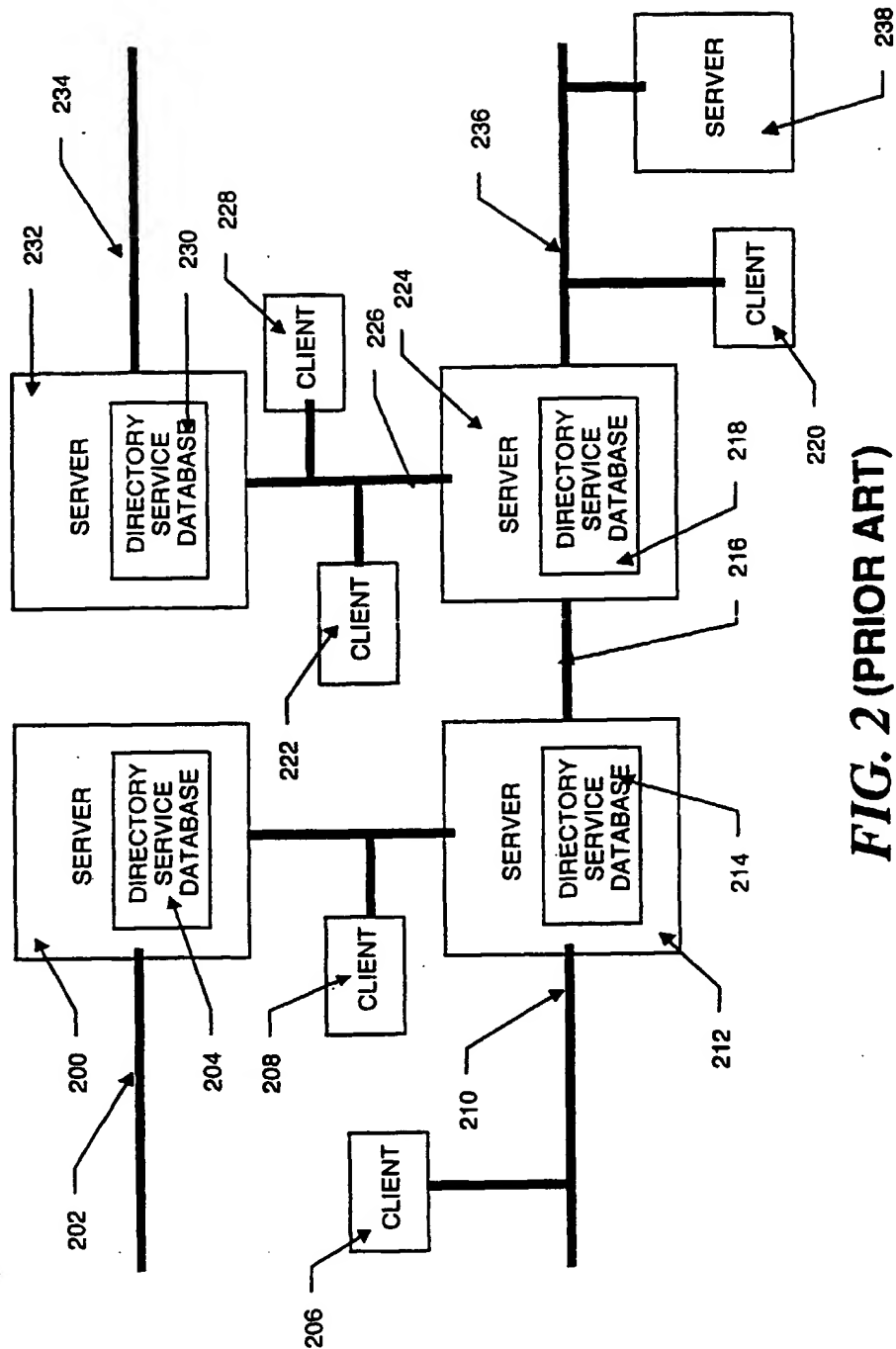


FIG. 2 (PRIOR ART)

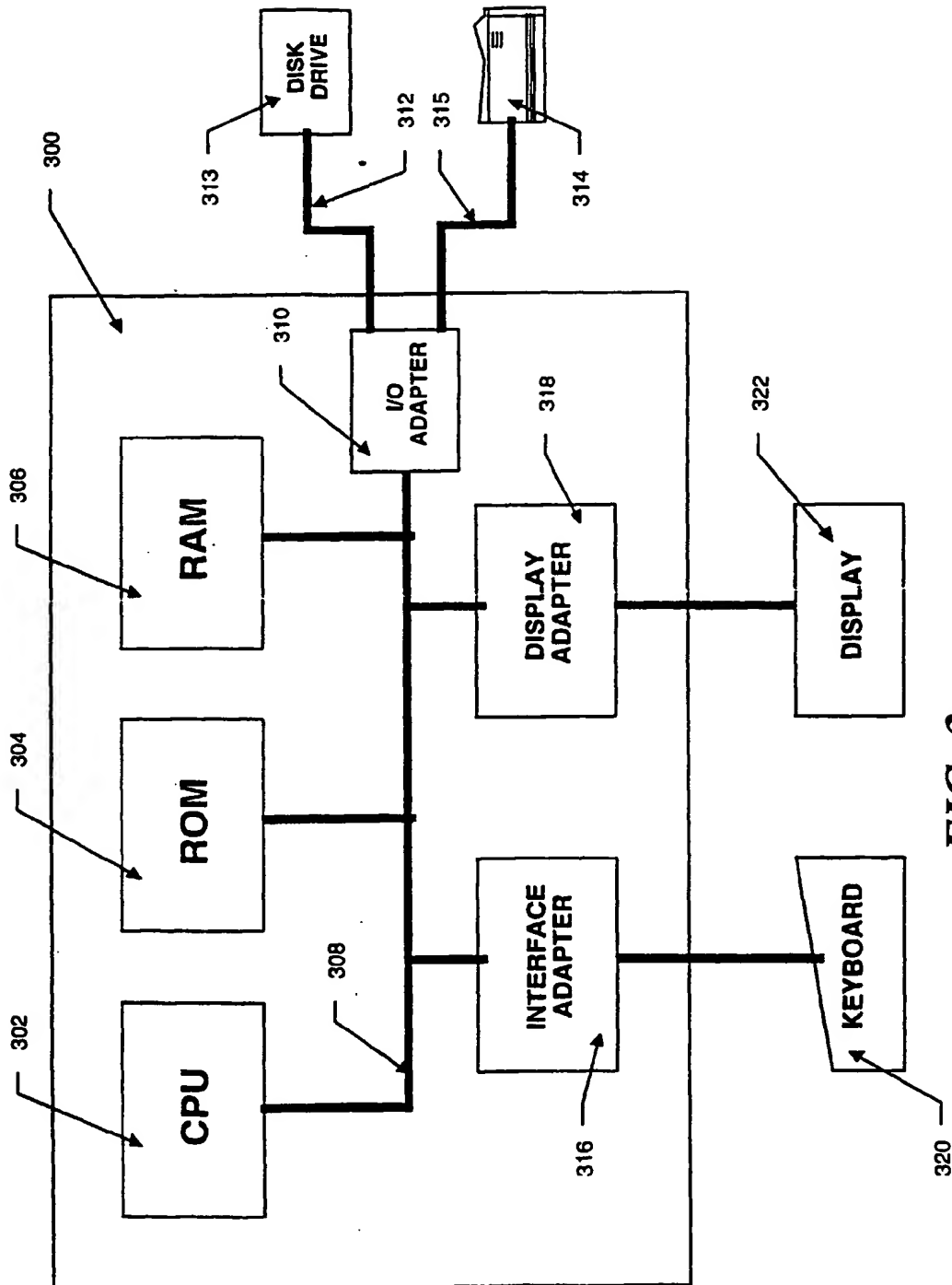


FIG. 3

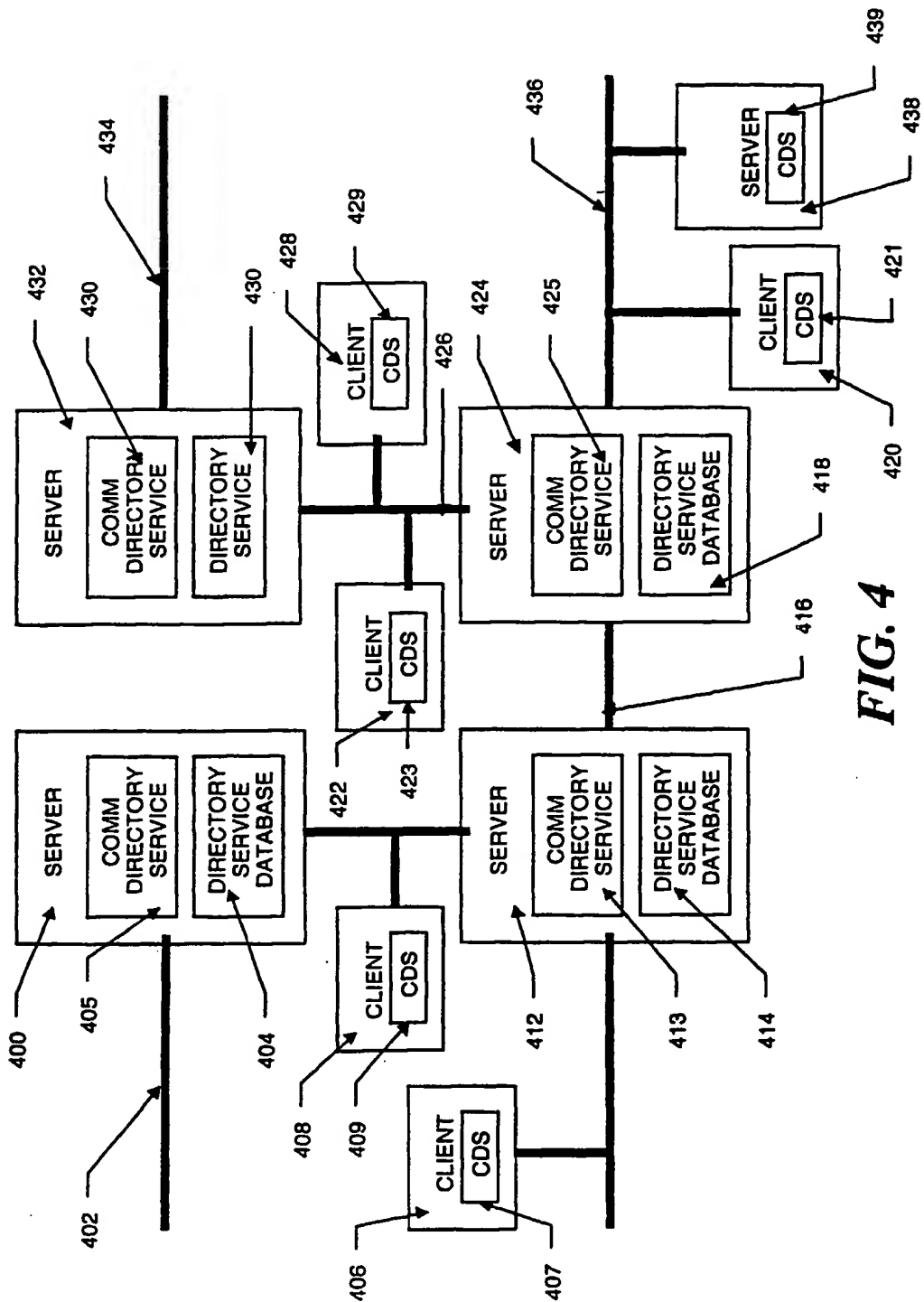


FIG. 4

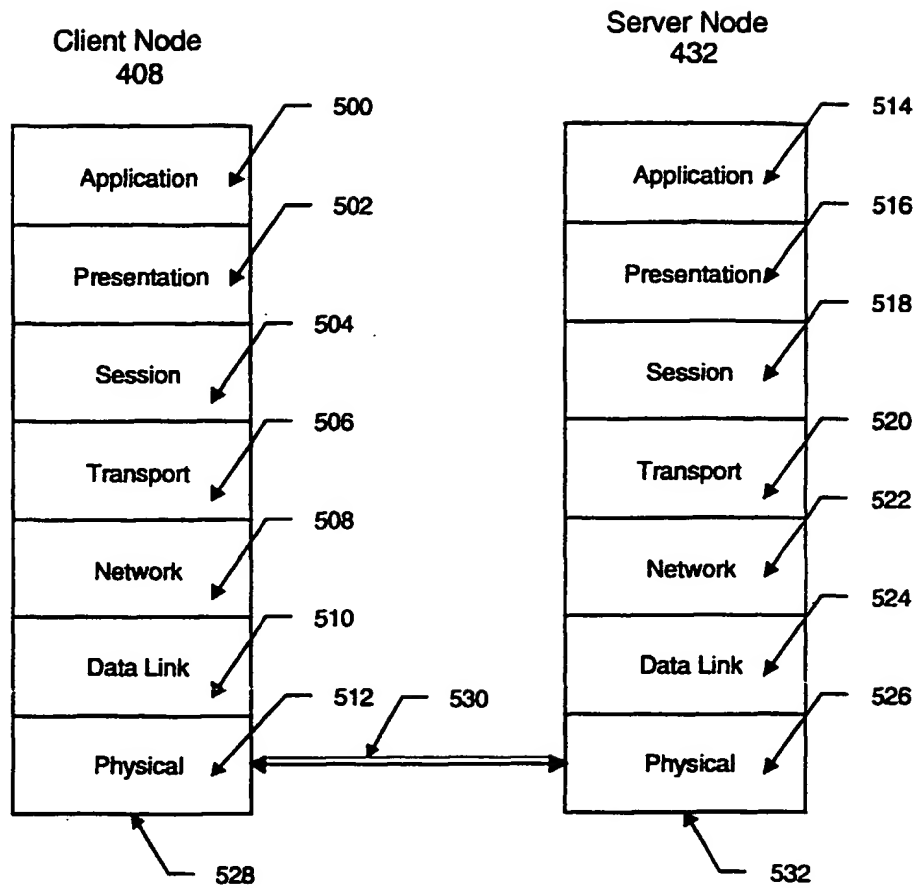


FIG. 5 (Prior Art)

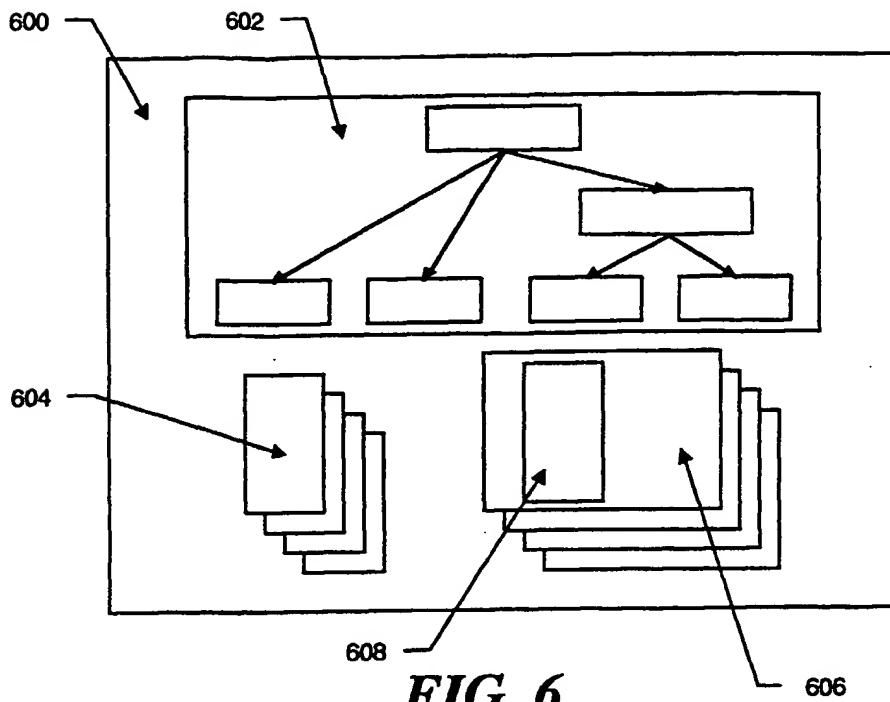


FIG. 6

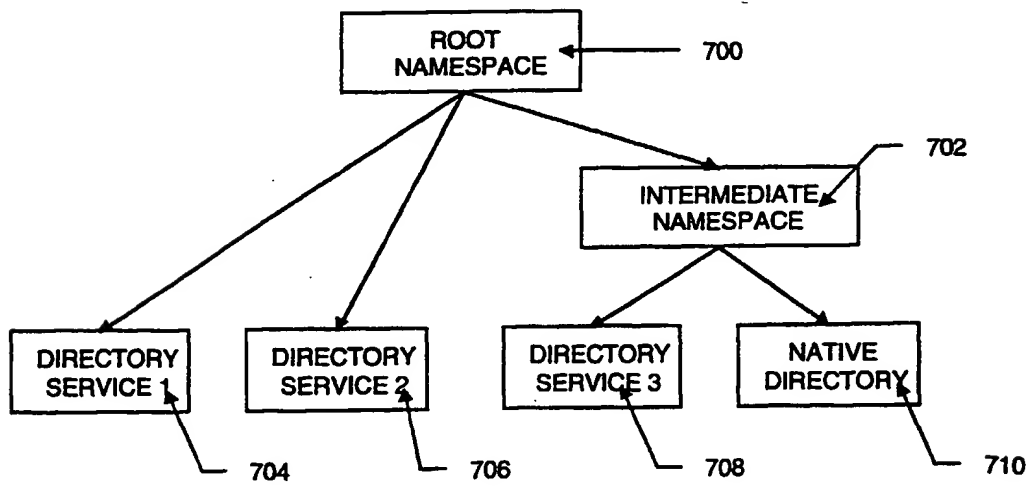


FIG. 7

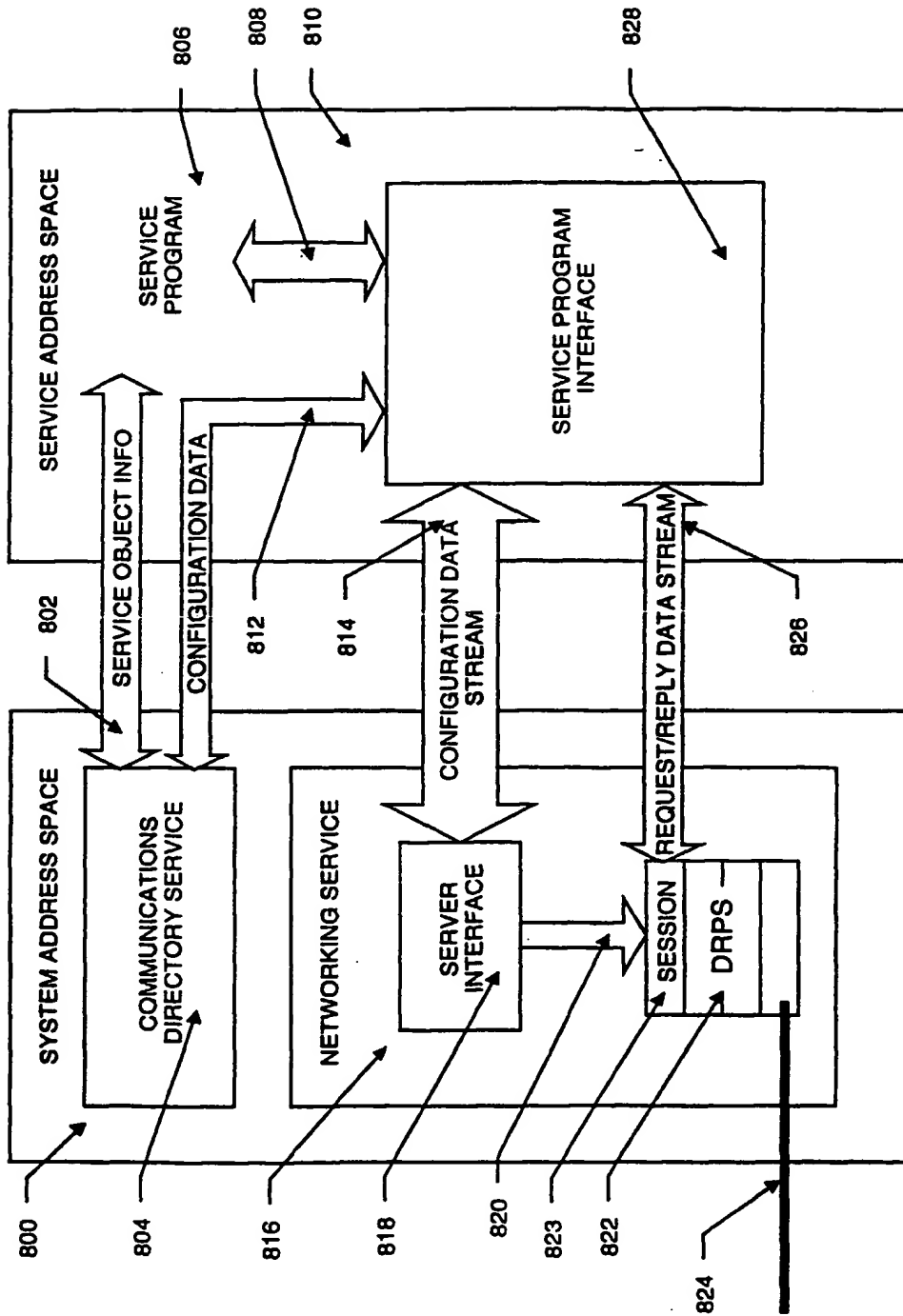
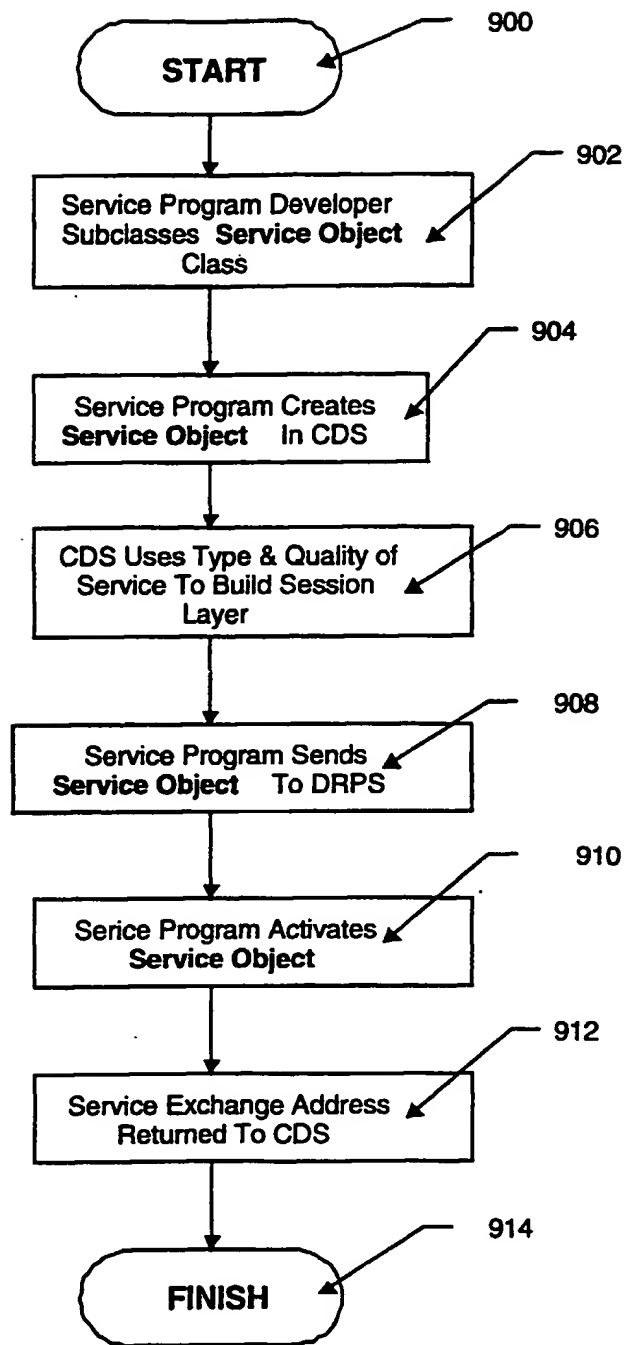
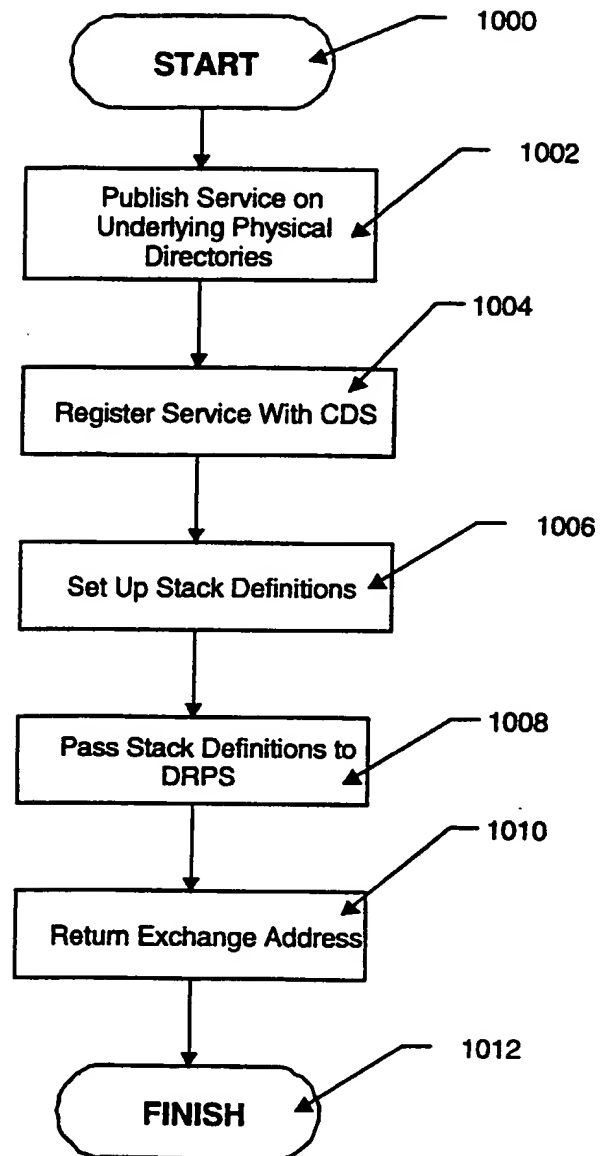


FIG. 8

**FIG. 9**

**FIG. 10**

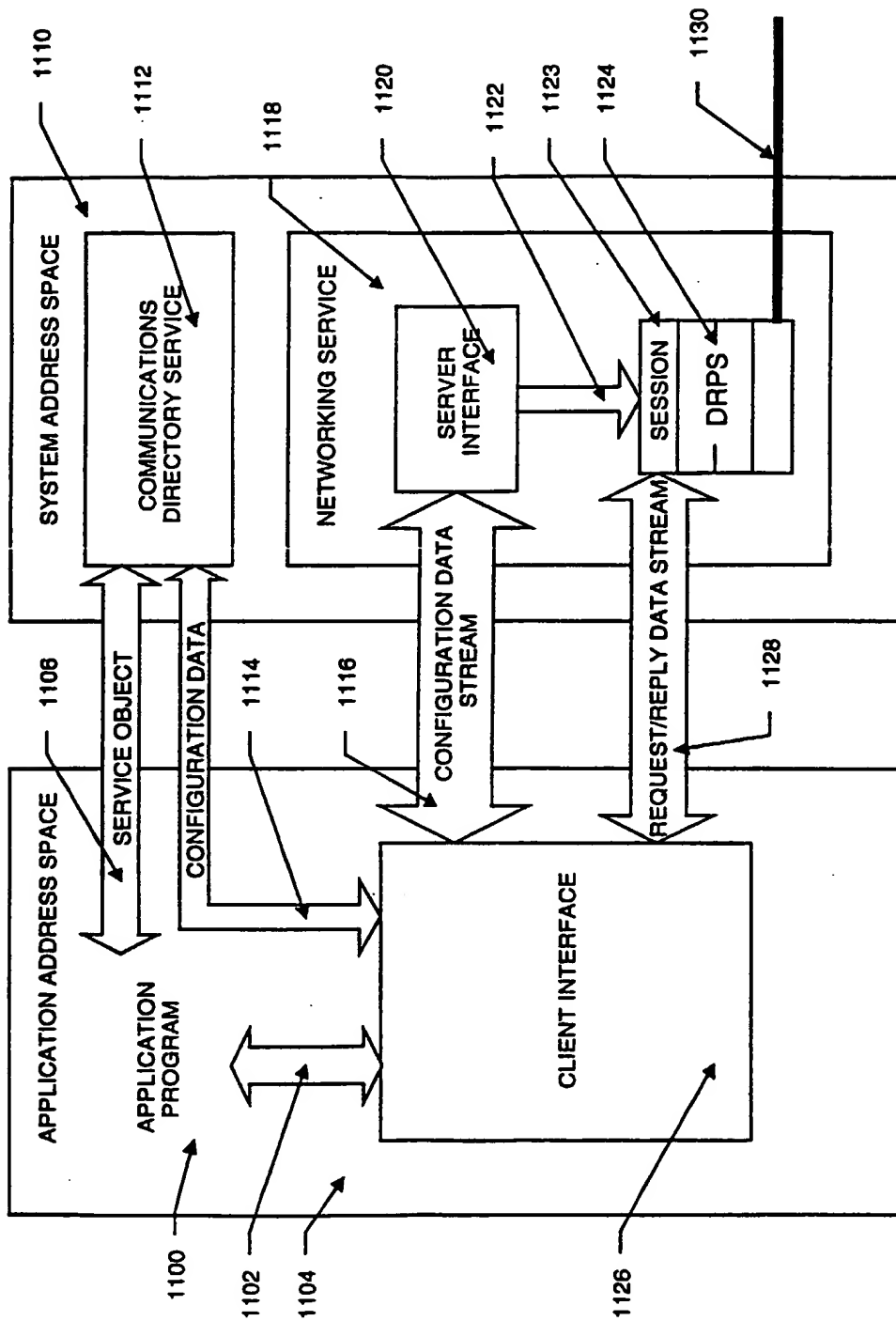
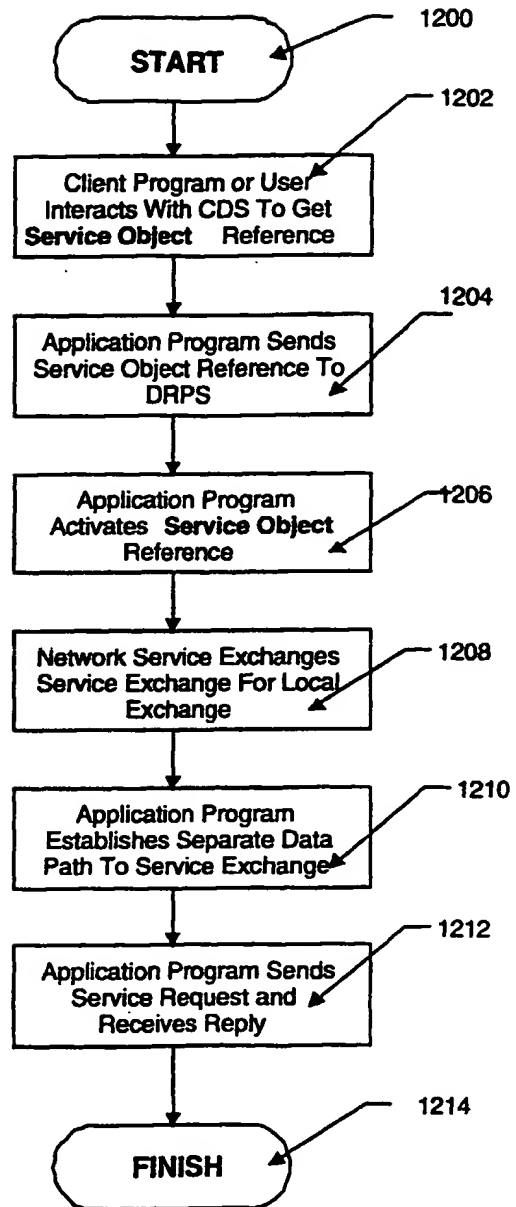
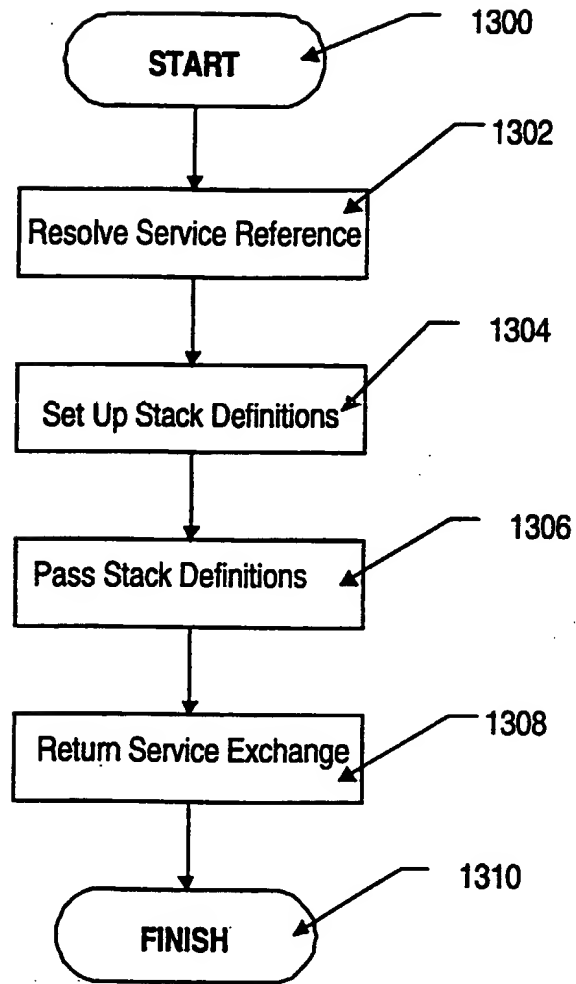


FIG. 11

**FIG. 12**

**FIG. 13**